МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Івано-Франківський національний технічний
університет нафти і газу

Кафедра комп’ютеризованого машинобудування

В. Б. Копей

МОВА ПРОГРАМУВАННЯ PYTHON ДЛЯ ІНЖЕНЕРІВ І НАУКОВЦІВ

НАВЧАЛЬНИЙ ПОСІБНИК

Івано-Франківськ
2019

УДК 004.43
К 65

Рецензент:

Панчук В. Г., доктор технічних наук, професор, завідувач кафедри комп'ютеризованого машинобудування Івано-Франківського національного технічного університету нафти і газу

К65 **Копей В. Б.** Мова програмування Python для інженерів і науковців: Навчальний посібник / В. Б. Копей - Івано-Франківськ: ІФНТУНГ, 2019. - 275 с.

Навчальний посібник містить приклади програм мовою Python з коментарями. Розглянуто основи програмування, стандартну бібліотеку та зовнішні бібліотеки для технічних та наукових обчислень. Розроблено відповідно до робочих програм дисциплін "Основи програмування" та "Об'єктно-орієнтоване програмування" для підготовки бакалаврів за спеціальністю 131 - Прикладна механіка.

УДК 004.43

ЗМІСТ

ВСТУП

Python - це популярна високорівнева мова програмування загального призначення з акцентом на продуктивність розробки. Python працює майже на усіх відомих платформах, є відкритим і вільним програмним забезпеченням, виконується шляхом інтерпретації байт-коду, підтримує кілька парадигм програмування (у тому числі об'єктно-орієнтоване), код програм компактний і легко читається (рис. 1). Мові характерні динамічна типізація, повна інтроспекція, зручні структури даних (кортежі, списки, словники, множини), велика стандартна бібліотека та велика кількість сторонніх бібліотек різноманітного призначення. Інтерпретатор Python має інтерактивний режим роботи, при якому введені з клавіатури оператори відразу ж виконуються, а результат виводиться на екран. Наприклад:

```
>>> a=1
>>> b=2
>>> a+b
3
>>>
```

Завдяки цим перевагам Python широко застосовується прикладними програмістами, зокрема інженерами і науковцями.

Основна мета цього посібника - швидке ознайомлення з основними можливостями Python для створення прикладного програмного забезпечення в галузі науки і техніки. Книга також може бути використана як довідник з Python і її пакетів. Посібник призначено для тих, хто уже володіє основами програмування якою-небудь алгоритмічною мовою. Паралельно з посібником автор рекомендує використовувати літературу [1-58] для глибшого освоєння матеріалу. Початківцям у першу чергу слід ознайомитись з книгами [16, 19, 21, 56, 58, 5] для вивчення основ Python та її стандартної бібліотеки.



Рисунок 1 - Переваги мови Python

Автор намагався продемонструвати максимум можливостей Python на мінімальному за обсягом коді, тому більшість прикладів є дещо штучними. Приклади програм містять коментарі, що надруковані курсивом після символу #. Ці коментарі не виконуються інтерпретатором. Код програм і результати їх виведення надруковані моноширинним шрифтом так:

```
код програми
```

```
текст виведення програми в консолі
```

Вихідний код усіх прикладів доступний для вільного завантаження на github.com (https://github.com/vkopey/Python-for-engineers-and-scientists). Цей код розмічений спеціальним чином і містить Markdown-текст в рядкових Python-літералах ("""…"""), що дозволяє генерувати з коду документи у форматах Jupyter Notebook, HTML та MS Word 2007. Детальніше про це - https://github.com/vkopey/py2nb.

В прикладах використовується версія Python 2.7. Ви можете завантажити інтерпретатор Python 2.7 з офіційного сайту (http://python.org), або один зі сторонніх дистрибутивів Python 2.7 (Anaconda [http://www.anaconda.com], WinPython [http://winpython.github.io], Python(x,y) [http://python-xy.github.io]), які містять велику кількість пакетів. Якщо потрібного пакету немає, то його можна установити за допомогою менеджера пакетів, наприклад так:

```
pip install <назва пакету>
```

Для аналізу прикладів та розв’язування задач зручно користуватись простим і невимогливим середовищем розробки Pyzo (http://www.pyzo.org), який має підказку коду з інтерактивним показом рядка документації. З електронної версії посібника ви можете копіювати приклади, які обведені рамкою, прямо в редактор коду. Під час копіювання дотримуйтесь правил відступів в Python. Один відступ складається з чотирьох пробілів, два - з восьми і т.д. Для уникнення проблем з кодуванням символів кожна програма повинна починатись з рядка:

```
# -*- coding: utf-8 -*-
```

Автор буде вдячний читачам за зроблені зауваження і побажання, які можна залишити на сайті проекту (https://github.com/vkopey/Python-for-engineers-and-scientists).

РОЗДІЛ 1. МОВА PYTHON ТА ЇЇ СТАНДАРТНА БІБЛІОТЕКА

Найпростіша програма

```
print "Hello World!" # вивести на екран "Hello
World!"
```

```
Hello World!
```

Програма для додавання двох чисел

Функція `input` чекає введення Python-виразу з консолі і повертає значення цього виразу. Для введення тільки рядків використовуйте функцію `raw_input`.

```
a = input("Введіть перше число: ") # ввести a
b = input("Введіть друге число: ") # ввести b
c=a+b # присвоїти c значення виразу a+b
print c # вивести на екран c
```

```
Введіть перше число: 2
Введіть друге число: a+1
5
```

Числові типи даних

До числових типів даних належать: цілі, дійсний, булевий і комплексний. В Python застосовується динамічна типізація - тип змінної визначається під час операції присвоювання. Тип змінної можна дізнатись за допомогою функції `type`.

```
a=16 # ціле десяткове int
b=020 # ціле вісімкове int
c=0x10 # ціле шістнадцяткове int
d=e=16L # довге ціле long
print a,b,c,d,e,type(e)
x=5.71 # дійсне float
```

```
y=-3.95e+3 # дійсне float
print x,y
i=True # булеве bool
print i
cn1=1+1j # комплексне complex
print cn1
```

```
16 16 16 16 16 <type 'long'>
5.71 -3950.0
True
(1+1j)
```

Оператори числових типів

Приклад показує використання найбільш уживаних операторів для числових типів. В складних виразах дотримуйтесь пріоритету операторів. Наприклад у виразі `1+x*2` спочатку виконується множення, а потім додавання. В наступному списку пріоритет операторів зменшується зверху вниз:

- `, [...] {...} `...`` - створення кортежу, списку, словника, конвертація рядка
- `s[i] s[i:j] s.attr f(...)` - індексування, зрізи, атрибути, виклик функції
- `+x -x ~x` - унарні оператори
- `x**y` – степінь
- `x*y x/y x%y` - множення, ділення, остача від ділення
- `x+y x-y` - додавання, віднімання
- `x<<y x>>y` - побітовий зсув
- `x&y` - побітове І
- `x^y` - побітове XOR (виключне АБО)
- `x|y` - побітове АБО
- `x<y x<=y x>y x>=y x==y x!=y x<>y` – порівняння
- `x is y   x is not y` – ідентичність
- `x in s   x not in s` – членство

- `not x` - булеве заперечення
- `x and y` - булеве І
- `x or y` - булеве АБО
- `lambda args: expr` - безіменна функція

```
a=int("7") # перетворення в ціле
b=long(9.7) # перетворення в довге ціле
x=float("3.14") # перетворення в дійсне
cn1=complex(1,1) # перетворення в комплексне
cn2=(1+2j)/cn1 # ділення комплексних чисел
m=abs(cn2) # модуль комплексного числа
i=bool(2>1) # перетворення в булеве
y=abs(-1.2) #модуль y=(-1.2).__abs__()
print a,b,x,cn2,cn2.real,cn2.imag,m,i,y
z=(-x*y+1)/y**2 # вираз з операторами: унарний мінус,
множення, додавання, ділення, степінь
r=9/5 # ділення цілих r=(9).__div__(5)
u=9//5 # цілочисельне ділення
v=9%5 # остача від ділення
w=divmod(9,5) # кортеж 9//5, 9%5
j=2>1 and 1<=0 and not(1==1 or 1!=0 or False) #
логічний вираз
k=round(2.91754,2) # заокруглити до 2 знаків після
коми
# вивести допомогу по функції: help(round)
print z,r,u,v,w,j,k
print eval("a+b") # значення динамічно побудованого
виразу
exec r"print a+b" # виконти Python-код (див. також
execfile)
```

```
7 9 3.14 (1.5+0.5j) 1.5 0.5 1.58113883008 True 1.2
-1.92222222222 1 1 4 (1, 4) False 2.92
16
16
```

Оператор умови if

Інструкція `if` виконує певні команди тоді, коли значення логічного виразу рівне `True` (істина). Інструкція `if` може застосовуватись з `elif` та/або `else`. Якщо значення логічного виразу після `if` рівне `False` (не істина), то виконується інструкція `elif`, або, якщо її немає, виконується `else`. Послідовних інструкцій `elif` може бути довільна кількість.

```
x = 2 # присвоїти x 2
if x<0: # якщо x<0 то
    y=1 # присвоїти y 1
    print "x<0, y=",y # вивести на екран
elif x>1 or x==0: # інакше, якщо x>0 або x=0 то
    y=2 # присвоїти y 2
    print "x>0, y=",y # вивести на екран
else: # інакше
    y=0 # присвоїти y 0
    print "x=0, y=",y # вивести на екран
```

```
x>0, y= 2
```

Оператор циклу for

Інструкція `for` повторює виконання певних команд, для кожного елемента послідовності.

```
for x in range(0,11): # для x в діапазоні [0,11)
    print x**2, # вивести на екран квадрат x
```

```
0 1 4 9 16 25 36 49 64 81 100
```

Оператор циклу while

Інструкція `while` повторює виконання певних команд, поки значення логічного виразу рівне `True`.

```
x=0 # присвоїти x 0
while x<=10: # поки x менше рівне 10
    print x**2, # вивести на екран квадрат x
    x=x+1 # збільшити x
```

```
0 1 4 9 16 25 36 49 64 81 100
```

Оператори break і continue

Інструкція `break` негайно завершує виконання циклу, а інструкція `continue` негайно переходить до наступної ітерації циклу.

```
x=0 # присвоїти x 0
while x<=10: # поки x менше рівне 10
    y=x**2
    x+=1 # збільшити x
    if y>50: # якщо y>50
        break # перервати цикл
    elif 10<y<30: # якщо 10<y<30
        print "*", # вивести на екран "*"
        continue # перейти до наступної ітерації
    print y, # вивести на екран y
```

```
0 1 4 9 * * 36 49
```

Послідовність кортеж. Оператори для усіх послідовностей

Кортеж - це об'єкт-контейнер типу `tuple`, який містить незмінну послідовність елементів довільного типу. Нижче показані спільні для усіх послідовностей (tuple, list, str, unicode) оператори.

```
a = (1,2,3,4,5) # кортеж
b = "a",5,3.07 # кортеж з різнотипними елементами
x,y=3,5 # використання кортежу в операторі
присвоювання
print a,len(a),b,len(b) # вивести кортежі і їх
```

```
довжини
# вивести елементи [індекс першого:індекс
останнього:крок]
print a[0],a[-1],a[1:4:2],a[::2]
print min(b),max(b) # мінімальне і максимальне
print a+b # об'єднання
print 2*a # об'єднані 2 копії
print "a" in b # чи "a" належить b
for x in a: # для кожного x у a
    print x, # вивести x
```

```
(1, 2, 3, 4, 5) 5 ('a', 5, 3.07) 3
1 5 (2, 4) (1, 3, 5)
3.07 a
(1, 2, 3, 4, 5, 'a', 5, 3.07)
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
True
1 2 3 4 5
```

Послідовність рядок

Рядок - це об’єкт-контейнер типу `str`, який містить незмінну послідовність символів. Для створення літерала рядка його потрібно взяти в апострофи, лапки або потрійні апострофи чи лапки. Кодування символів літерала рядка відповідає кодуванню символів файлу програми (тут це UTF-8). Кодування символів (ASCII, UTF-8, CP1251, CP866 та ін.) - це таблиця, у якій кожен символ кодується одним або більшою кількістю байтів.

```
s1="Рядок1" # рядок
s2='"String"2\n' # рядок з спецсимволом переводу
рядка
s3='''Str
ing3''' # рядок з переводом рядка
s4=r"String\t4\n" # необроблюваний рядок
s5=u"String5" # Unicode рядок
```

```
s6=ur"String6\n" # Unicode необроблюваний рядок
print s1,s2,s3,s4,s5,s6,len(s6) # вивести рядки і довжину рядка s6
print s1+" "*5+str(3.14)+" "+repr(3.14) # об'єднання рядків
print s1[2],s1[:2],s1[2:],s1[-2] # зрізи рядка
print "я" in s1 # чи "я" належить s1
for x in s1: pass # для кожного символу x у s1 виконати пусту команду
print "x= %*.*f%s"%(5,2,51.2935,"mm") # форматування рядка
print "x= %4d %o %x"%(16,16,16) # форматування рядка
print "x= {0} {unit}".format('five', unit='mm') # форматування рядка
print "   hello  ".strip() # видалити пробіли на початку і в кінці
print "hello world!".find("ll",0,4) # знайти входження підрядка на проміжку
print "hello".replace("h","H",1) # замінити 1 раз "h" на "H"
lst="a,b,c".split(",") # розбити рядок на список (розділювач ",")
s1="" # пустий рядок
print s1.join(lst) # об'єднати список в рядок
print oct(16), hex(16) # вивести рядкове представлення вісімкового і шістнадцяткового числа
print ord('A'),chr(65),unichr(65),unicode('ABC','utf-8') # вивести код символу, символ за кодом, символ Unicode за кодом, об'єкт в заданому кодуванні
```

```
Рядок1 "String"2
Str
ing3 String\t4\n String5 String6\n 9
Рядок1     3.14 3.14
'\xd1' Р ядок1 '\xba'
```

```
True
x= 51.29mm
x=   16 20 10
x= five mm
hello
2
Hello
abc
020 0x10
65 A A ABC
```

Юнікод-рядки

Юнікод-рядок (unicode) - це послідовність символів Юнікоду. Для створення літерала юнікод-рядка перед лапками потрібно поставити символ `u`. Рядки unicode і str підтримують однакові операції.

```
s='звичайний рядок'
us=u'юнікод-рядок'
print type(s),isinstance(s, str) # <type 'str'> True
print type(us),isinstance(us, unicode) # <type 
'unicode'> True
s2=us.encode('utf-8') # кодує в звичайний рядок utf-8
us2=s.decode('utf-8') # декодує звичайний рядок utf-8 
в юнікод-рядок
us2=unicode(s, 'utf-8') # або так
print type(s2) # <type 'str'>
print type(us2) # <type 'unicode'>
print len('рядок') # 10 , бо кодується utf-8 (символи 
кодуються 1-6 байтами)
print len(u'рядок') # 5
```

Юнікод-літерали в Python 2

В Python 2 використовувати юнікод-літерали без застосування символу u можна так:

```
from __future__ import unicode_literals
print type('текст') # <type 'unicode'>, а не <type
'str'>
print type(u'текст') # <type 'unicode'>
```

Послідовність список

Список - це об'єкт-контейнер типу `list`, який містить послідовність елементів довільного типу. Ця послідовність може змінюватися.

```
a = [1,2,3,4,5] # список
b=[1.0/x for x in range(1,4) if x!=2] # генератор
списку
c=list("string") # список з рядка
d=range(5) # список з прогресії
e=[1,"abc",2.3,(1+1j),a] # список з різнотипними
елементами
print a,len(a),b,len(b) # вивести списки і їх довжини
print e[4] # вивести елемент з індексом 4 (список a)
print a[0],a[-1],a[1:4:2],a[::2] # вивести елементи
[індекс першого:індекс останнього:крок]
print min(a),max(a) # мінімальне і максимальне
print a+[6,7] # об'єднання
print 2*a # об'єднані 2 копії
print 3 in a # чи 3 належить a
a[0]=2 # присвоїти значення елементу з індексом 0
a[::2]=[0,0,0] # непарним елементам присвоїти 0
del a[4] # видалити елемент з індексом 4
a.append(5) # додати в кінець 5
a.extend([6,7]) # розширити список
```

```
print a.count(0) # кількість елементів рівних 0
print a.index(5) # мінімальний індекс елемента зі
значенням 5
a.insert(1,9) # вставити в позицію 1
print a.pop(1) # вивести елемент з індексом 1 і
видалити
a.reverse() # обернути список
a.sort() # сортувати за зростанням
for x in a[:]: # для кожного x у копії a
    print x, # вивести x
```

```
[1, 2, 3, 4, 5] 5 [1.0, 0.3333333333333333] 2
[1, 2, 3, 4, 5]
1 5 [2, 4] [1, 3, 5]
1 5
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
True
2
4
9
0 0 2 4 5 6 7
```

Словник. Оператори для словників

Словник (dict) - це асоціативний масив, який містить сукупність пар (ключ, значення). Значеннями можуть бути об'єкти будь-якого типу, а ключами - об'єкти, які не змінюються (числа, рядки, кортежі).

```
d1={1:"Іванов",2:"Петров",3:"Коваль"} # словник
d2=dict([("Іванов",1982),("Петров",1980),("Коваль",19
78)]) # словник
d3=dict(x=1.5,y=5.2,z=3.0) # словник
d2["Іванов"]=1983 # зміна значення за ключем "Іванов"
del d2["Іванов"] # видалити елемент за ключем
```

```
print "Петров" in d2 # чи є ключ "Петров" у словнику d2
for k,v in d2.iteritems():print k,v, # цикл за елементами
print
for k in d2.iterkeys():pass # цикл за ключами
for v in d2.itervalues():pass # цикл за значеннями
l3=d2.items() # список з елементів-пар (ключ,значення)
l1=d2.keys() # список з ключів
l2=d2.values() # список зі значень
print d2.get("Петров",0) # значення за ключем, якщо немає - 0
d4=d1.fromkeys([1,2,3],"Male") # новий словник з ключів d1
print d4 # вивести словник
```

```
True
Коваль 1978 Петров 1980
1980
{1: 'Male', 2: 'Male', 3: 'Male'}
```

Множина

Множина - це об’єкт-контейнер типу `set`, який містить невпорядковану сукупність елементів, які не повторюються.

```
s1={1,1,2,3,2,5,3,1,5} # множина
s2=set([7,7,8,9,1,2]) # множина зі списку
s3=set("hello") # множина з рядка
print s1,len(s1),s2,len(s2) # вивести множини і їх довжини
print 2 not in s1 # чи 2 не в множині s1?
for x in s1: pass # для кожного в множині виконати пусту команду
s1.add(7) # додати елемент
```

```
s1.remove(7) # видалити елемент
s1.discard(7) # видалити елемент, якщо він є
print s1.pop() # вивести довільний і видалити його
print s1.issubset(s2) # чи кожен у s1 є у s2 (s1 <=
s2)
print s1.issuperset(s2) # чи кожен у s2 є у s1 (s1 >=
s2)
print s1.union(s2) # об'єднання (s1 | s2)
print s1.intersection(s2) # перетин (s1 & s2)
print s1.difference(s2) # різниця (s1 - s2)
print s1.symmetric_difference(s2) # симетрична
різниця (s1 ^ s2)
```

```
set([1, 2, 3, 5]) 4 set([8, 9, 2, 1, 7]) 5
False
1
False
False
set([1, 2, 3, 5, 7, 8, 9])
set([2])
set([3, 5])
set([1, 3, 5, 7, 8, 9])
```

Функції

Функція - це частина коду програми (підпрограма), до якого можна звернутись з інших місць програми. Визначити функцію можна за допомогою інструкції `def`. Функція може повертати певне значення за допомогою інструкції `return`. Атрибут `__doc__` містить документацію функції.

```
def sum(a,b=0): # визначити функцію з параметрами a,
b
    "Повертає суму двох чисел" # рядок документації
    c=a+b
    return c # функція повертає значення c
```

```
print sum(2,2), # вивести значення функції (a=2, b=2)
print sum(a=2,b=2), # або так
print sum(2) # вивести значення функції (a=2, b=0)
print sum.__doc__ # вивести документацію
```

```
4 4 2
Повертає суму двох чисел
```

Функції з довільним числом аргументів

Наступна функція має один обов'язковий аргумент `a`, довільну кількість неіменованих аргументів `*b`, і довільну кількість іменованих аргументів `**c`.

```
def f(a,*b,**c):
    print a,b,c # тут b - кортеж, c - словник
f(1,2,3,x=4,y=5) # a=1 b=(2, 3) c={'y': 5, 'x': 4}
f(a=1,b=2,c=3,x=4,y=5) # a=1 b=() c={'y': 5, 'x': 4,
'c': 3, 'b': 2}
```

```
1 (2, 3) {'y': 5, 'x': 4}
1 () {'y': 5, 'x': 4, 'c': 3, 'b': 2}
```

lambda-функції

Невеликі анонімні функції у вигляді одного виразу можуть бути описані за допомогою ключового слова `lambda`.

```
f = lambda x, y: x + y # визначення функції за
допомогою виразу
print f(2,3) # вивести значення функції (x=2,y=3)
```

```
5
```

Рекурсивні функції

Рекурсивна функція викликає саму себе. Існує обмеження на глибину рекурсії.

```
def bin(n): # рекурсивна функція
    "Повертає список з двійковим поданням числа"
    if n == 0: return [] # якщо n == 0, повертає
пустий список
    n, d = divmod(n, 2) #або n=n//2; d=n%2
    return bin(n) + [d] # повертає bin(n) + [d]
print bin(5) # виклик функції
```

```
[1, 0, 1]
```

Замикання

Замикання (closure) — це функція, яка визначена в тілі іншої функції і в якій є посилання на змінні, що оголошені зовні.

```
a=0 # глобальна змінна
def func(b): # функція
    def fn(c): # внутрішня функція (замикання)
        global a # звертання до глобальної змінної
        a=1 # зміна значення глобальної змінної
        print a,b,c # вивести a,b,c
    print 'func'
    return fn # повернути функцію
f=func(2) # аргумент b=2, виведе: 'func'
f(3) # аргумент c=3, виведе: 1 2 3
print a # 1 - значення глобальної змінної змінилось
```

```
func
1 2 3
1
```

Обробка виняткових ситуацій

Інструкції `try` і `except` дозволяють перехоплювати і обробляти виняткові ситуації - помилки, що виникають під час виконання програми. Якщо помилка виникає в блоці `try`, то

керування передається тому блоку `except`, який відповідає типу помилки.

```
import sys # імпорт модуля sys
for x in -1, 0, "0.2", 1e1000: # повторити наступні команди для різних значень змінної x
    try: # перехоплювати помилки виконання
        assert x>=0 # якщо x менше 0, генерувати AssertionError
        y=int(x) # помилка ValueError
        z=1/x # помилка ZeroDivisionError
    except ZeroDivisionError: # якщо ділення на нуль
        print x,"Помилка! Ділення на нуль"
    except: # якщо інші помилки
        print x,sys.exc_type # вивести тип помилки
```

```
-1 <type 'exceptions.AssertionError'>
0 Помилка! Ділення на нуль
0.2 <type 'exceptions.ValueError'>
inf <type 'exceptions.OverflowError'>
```

Файли

Файл - це інформаційний об’єкт, який містить послідовність байтів і розміщений у файловій системі на носію інформації. Усі файли є бінарними, але якщо для файлу застосовується кодування символів (ASCII, UTF-8, CP1251 або інше), то його називають текстовим. Наприклад файл з кодом Python програми (.py) є текстовим. В бінарних файлах кодування символів не застосовується. Для роботи з файлом його відкривають функцією `open`, яка створює файловий об’єкт, що має методи запису, читання і закриття файлу.

```
f1=open("file1.txt", "w") # відкрити текстовий файл для запису
f1.write("Line1\n") # записати рядок ('\n' - символ
```

```
кінця рядка)
f1.close() # закрити файл

f2=open("file1.txt", "a") # відкрити текстовий файл
для добавлення
f2.writelines(("Line2\n","Line3\n")) # записати
послідовність рядків
f2.close() # закрити файл

f3=open("file1.txt", "r+") # відкрити текстовий файл
для читання і запису
print f3.read() # читати весь файл
f3.seek(0) # установити позицію на початок файлу
print f3.readline(),f3.tell() # читати рядок, вивести
поточну позицію
print f3.readlines() # читати список рядків до кінця
f3.seek(0) # установити позицію на початок файлу
for line in f3: # для кожного рядку у файлі
    pass # виконати пусту команду
f3.close() # закрити файл

f4=open("file1.txt", "rb") # відкрити бінарний файл
для читання
# спробуйте також відкрити цей файл як текстовий "r"
f4.seek(7) # установити позицію після байта 7
while True: # читає файл побайтово
    b=f4.read(1) # читати байт
    if not b: break # перервати цикл, якщо байтів
немає
    print ord(b), # числове подання Юнікод-символу
#зверніть увагу на два байти (13 10), які в текстових
файлах Windows використовуються для позначення кінця
рядка
f4.close() # закрити файл
```

```
Line1
Line2
Line3

Line1
7
['Line2\n', 'Line3\n']
76 105 110 101 50 13 10 76 105 110 101 51 13 10
```

Модулі

Модулем є будь-який файл з вихідним кодом Python. Команда `import` імпортує модуль в програму, тобто код модуля виконується в окремому просторі імен і створюється об’єкт модуля, який містить об’єкти з цього простору імен. Команда `from` працює аналогічно, але імпортує тільки визначені імена. Пакети містять кілька модулів. Щоб створити пакет, створіть папку з його іменем і розмістіть в ній файл `__init__.py`. Він виконується під час імпорту пакета. Для прикладу створіть нову папку проекту `c:\1`. В ній створіть файли `main.py`, `module1.py` та папку `package1`. В папці `package1` створіть файли `__init__.py`, `module1.py`, `module2.py`. Виконайте модуль `main.py`.

Файл c:\1\main.py:

```
import sys # імпортує модуль (і тільки 1 раз)
import module1 as m # імпортувати модуль і змінити його ім'я на m
from package1 import * # імпортувати з пакету все
#або
#from package1.module1 import * # імпортувати з модуля все
#from package1.module2 import * # імпортувати з модуля все
if __name__ == '__main__':# якщо модуль виконується, а не імпортується
```

```
    print __name__, __file__ # ім'я і файл модуля
    #print sys.path # шляхи пошуку модулів
    sys.path.append("c:\SomeFolder") # додати шлях
пошуку модулів
    print m.__doc__ # рядок документації модуля
    print m.__dict__.keys() # список імен
    m.a='a_'
    m.f() # виведе 'a_'
    print a,b,c
    reload(m) # повторно завантажити модуль
    print m.a # виведе 'a'
```

```
module1 C:\1\module1.pyc
package1 C:\1\package1\__init__.pyc
package1.module1 C:\1\package1\module1.pyc
package1.module2 C:\1\package1\module2.pyc
a
b
__main__ C:\1\main.py
module1 doc
['a', 'f', '__builtins__', '__file__', '__package__',
'__name__', '__doc__']
a_
a b c
module1 C:\1\module1.pyc
a
```

Файл c:\1\module1.py:

```
'''module1 doc''' # рядок документації модуля
print __name__, __file__
a='a' # атрибут модуля
def f(): # атрибут модуля
    print a
```

Файл c:\1\package1__init__.py:

```
# цей файл виконується під час імпорту пакета
print __name__, __file__
from module1 import *
from module2 import *
```

Файл c:\1\package1\module1.py:

```
print __name__, __file__
a='a' # атрибут модуля
from module2 import b # з модуля копіювати b
# зверніть увагу на порядок інструкцій і будьте уважні з рекурсивним імпортом
print b
```

Файл c:\1\package1\module2.py:

```
print __name__, __file__
b='b' # атрибут модуля
c='c' # атрибут модуля
d='d' # не буде імпортуватись інструкцією from *
_e='_e' # не буде імпортуватись інструкцією from *
from module1 import * # зверніть увагу на порядок інструкцій
print a
__all__=[b,c] # список імен, які імпортуються інструкцією from *
```

Математичні функції

Стандартний модуль math містить математичні функції і константи. Наприклад, якщо модуль імпортується так: `import math`, то функції потрібно викликати так: `math.sin(x)`.

```
from math import * # імпортувати усе з модуля math
x,y=1,1
acos(x) # арккосинус x
asin(x) # арксинус x
atan(x) # арктангенс x
atan2(y,x) # atan(y/x)
ceil(x) # найменше ціле, більше або рівне x
cos(x) # косинус x
cosh(x) # гіперболічний косинус x
e # константа e
exp(x) # експонента (e**x)
fabs(x) # абсолютне значення x
floor(x) # найбільше ціле, менше або рівне x
fmod(x,y) # остача від ділення x на y
frexp(x) # мантиса і порядок x як пара (m, i), де m - число з плаваючою комою, а i - ціле, таке, що x = m * 2.**i. Якщо x=0, то повертає (0,0), інакше 0.5 <= abs(m) < 1.0
hypot(x,y) # sqrt(x*x + y*y)
ldexp(x,y) # x * (2**y)
log(x) # натуральний логарифм x
log10(x) # десятковий логарифм x
modf(x) # пара (y,q) - ціла та дробова частина x. Обидві частини мають знак x
pi # константа пі
pow(x,y) # x в степені y (або x**y)
sin(x) # синус x
sinh(x) # гіперболічний синус x
sqrt(x) # корінь квадратний від x
tan(x) # тангенс x
tanh(x) # гіперболічний тангенс x
```

```
0.7615941559557649
```

Вбудовані функції для роботи з послідовностями

Для полегшення роботи з послідовностями існують вбудовані функції:

- `filter` - фільтрує послідовність за допомогою заданої функції;
- `map` - застосовує функцію для кожного елемента;
- `reduce` - застосовує до елементів функцію двох аргументів кумулятивно зліва направо;
- `zip` - об’єднує послідовності в список кортежів;
- `enumerate` - генерує пронумеровану послідовність;
- `sorted` - сортує послідовність.

```
a=[1,2,3,4,5] # список
def fn1(x): return x!=3 # функція повертає істину,
якщо x не 3
print filter(fn1,a) # відфільтрований функцією fn1
список
#print filter(lambda x:x!=3,a) # або так
#print [x for x in a if x!=3] # або так
def fn2(x): return x**2 # функція повертає квадрат
числа
print map(fn2,a) # застосовує fn2 до кожного елемента
def fn3(x,y): return x+y # функція повертає суму двох
чисел
print map(fn3,a,[1,2,3,4,5]) # сума списків
#print map(lambda x,y:x+y,a,[1,2,3,4,5]) # або так
#print [x+y for x,y in zip(a,[1,2,3,4,5])] # або так
print reduce(fn3,a) # підрахувати (((1+2)+3)+4)+5
#print reduce(lambda x, y: x + y, a) # або так
print zip([1,2,3],[4,5,6]) # об'єднати списки
print [i for i in enumerate("abc")] # пронумерований
список
```

```
print sorted([(1,2),(2,1),(3,3)], key=lambda x: x[1])
# сортувати за елементами з індексом 1
```

```
[1, 2, 4, 5]
[1, 4, 9, 16, 25]
[2, 4, 6, 8, 10]
15
[(1, 4), (2, 5), (3, 6)]
[(0, 'a'), (1, 'b'), (2, 'c')]
[(2, 1), (1, 2), (3, 3)]
```

Генератори

Генератор створюється функцією, яка використовує інструкцію `yield`. Застосовуються для генерування послідовностей. Виклик метода `next` генератора виконує цю функцію, поки не досягнуто наступної інструкції `yield`, повертає значення після інструкції `yield` і зупиняє виконання функції. Наступний виклик метода `next` продовжує виконання функції з наступної за `yield` інструкції.

```
def generator(n): # функція генератора (генерує
послідовність)
    while n<3:
        yield n # генерувати значення n
        n+=1
genObj=generator(0) # створити об'єкт генератора,
який має можливість ітерації (є ітератором)
print genObj.next(), genObj.next(), genObj.next() # 0
1 2
#print genObj.next() # помилка: StopIteration
for x in generator(0): # або так
    print x, # виведе: 0 1 2
print
def generator2(n): # функція генератора з кількома
інструкціями yield
    while n<3:
```

```
        yield n # генерувати значення n
        yield n # генерувати значення n
        n+=1
for x in generator2(0):
    print x, # виведе: 0 0 1 1 2 2
```

```
0 1 2
0 1 2
0 0 1 1 2 2
```

Співпрограми

Співпрогама - це функція генератора, яка містить вираз (`yield`). Метод `send` об'єкта співпрограми передає їй дані, які повертаються виразом (`yield`).

```
def coroutine(): # співпрограма обробляє
послідовність вхідних даних
    while True:
        x=(yield) # отримати дані через send()
        yield 2*x # генерувати значення 2*x
corObj=coroutine() # створити об'єкт генератора
for x in [1,2,3]: # виведе: 2 4 6
    corObj.next() # переміститись до першої
інструкції (yield)
    print corObj.send(x), # передати дані
співпрограмі через вираз (yield), отримати значення
після другої інструкції yield
corObj.close() # завершити роботу з corObj
```

```
2 4 6
```

Ітератори

Ітератор - це об'єкт, який призначений для обходу елементів певного контейнера. Ітератори використовуються в інструкції `for`. Ітератор реалізує метод `next`, який повертає наступний елемент

контейнера, або викликає виняткову ситуацію `StopIteration`, якщо елементів більше немає.

```
it=iter([1,2,3]) # ітератор
for i in it: print i, # вивести 1 2 3
print
def f(l=[]): # функція
    l.append(1) # додати в список 1
    return l # повертає список
it=iter(f,[1,1,1]) # ітератор, який викликає f поки
нею не буде повернуто [1,1,1]
for i in it: print i,
```

```
1 2 3
[1] [1, 1]
```

Об'єкти

Python володіє потужними об'єктно-орієнтованими можливостями. Наприклад, усі змінні, функції і класи є об'єктам і володіють атрибутами і методами.

```
a=1 # створити змінну (об'єкт класу int) і присвоїти
їй 1
print a.__class__ # атрибут __class__ (клас об'єкта)
print a.__class__.__name__ # тип також є об'єктом і
має атрибут __name__ (ім'я)
b=a.__add__(2) # метод __add__ повертає суму a+2
b=a+2 # або так
x=a.__float__() # метод повертає дійсне число
x=float(a)
print a.__str__() # метод повертає рядок str(a)
```

```
<type 'int'>
int
1
```

Класи

Об’єктно-орієнтоване програмування (ООП) основане на використанні об’єктів, які є абстрактними моделями реальних об’єктів. Об’єкти створюються за допомогою спеціальних типів даних - класів. Кожен клас описує множину об’єктів певного типу. Основними принципами ООП є інкапсуляція, успадкування і поліморфізм. Інкапсуляція - об’єднання даних (атрибутів) і функцій їх опрацювання (методів) в класі. Наприклад, в класі `A` об’єднано атрибут `a` і метод `f`. Ідентифікатор `self` використовується в класах як посилання на об’єкт цього класу. Методи об’єктів повинні мати перший аргумент `self`.

```
class A: # визначення класу A
    a=5 # атрибут-дане a
    def f(self): # атрибут-метод f
        return self.a**2 # повертає квадрат a
obj=A() # створення об'єкта (екземпляра) obj класу A
obj.a=2 # зміна атрибута-даного a
print obj.f() # виклик методу f
```

```
4
```

Клас з конструктором

Конструктор - це спеціальний метод, який має назву `__init__` і викликається під час створення об’єкта. Часто використовується для ініціалізації атрибутів-даних об’єкта. В прикладі також показано можливість визначення методу поза класом.

```
def f2(self,a,x=1): # визначення методу поза класом
    self.a=a # присвоїти атрибуту-даному значення
аргументу a
    print self.f(x) # виклик методу f
class A: # визначення класу A
    "Клас A" # рядок документації
    a=0 # атрибут-дане
```

```
    def __init__(self,a): # конструктор
        self.a=a # присвоїти атрибуту-даному значення 
аргументу a
    def f(self,x): # атрибут-метод f
        return self.a**x # повертає a в степені x
    f2=f2 # атрибут-метод f2, визначений поза класом
obj=A(3) # створення об'єкта obj класу A, виклик 
конструктора (a=3)
obj.a=2 # зміна атрибута-даного a
print obj.f(2) # виклик методу f
print obj.__doc__ # вивести рядок документації
pow=obj.f # об'єкт-метод obj.f
print pow(3) # вивести значення pow(3)
obj.f2(5,2) # виклик методу f2
```

```
4
Клас A
8
25
```

Успадкування і поліморфізм

Успадкування - це можливість створення похідних класів шляхом успадкування ними членів базового класу. Наприклад, атрибут `a` класу `B` успадкований від класу `A`. Поліморфізм - здатність методів з однаковою специфікацією мати різну реалізацію. Наприклад, функції `f` класів `A` і `B` мають одну назву, але різну реалізацію.

```
class A(object): # визначення класу A, успадкованого 
від object
    a=0 # атрибут-дане
    def __init__(self,a): # конструктор
        self.a=a # присвоїти атрибуту значення 
аргументу a
    def f(self,x): # атрибут-метод f
```

```
        return self.a+x # повертає a+x
    def f2(self): # атрибут-метод f2
        return self.a+2 # повертає a+2
class B(A): # визначення класу B успадкованого від A
    b=0 # атрибут-дане
    def __init__(self,a,b): # конструктор
        A.__init__(self,a) # виклик конструктора
базового класу
        #super(B, self).__init__(a) # або так
        self.b=b #присвоїти атрибуту значення
аргументу b
    def f(self,x): # атрибут-метод f
        return self.b+x # повертає b+x
obj=B(1,2) # створення об'єкта obj класу B, виклик
конструктора (a=1,b=2)
print obj.a, obj.b # вивести значення атрибутів
print obj.f(2) # виклик методу f, описаного в класі B
print A.f(obj,2) # виклик методу f, описаного в класі
A
print obj.f2() # виклик методу f2
del obj # знищити об'єкт
#print obj.a # помилка! Об'єкта не існує
```

```
1 2
4
3
3
```

Атрибути класу і атрибути екземпляра

Важливо розрізняти атрибути класу і атрибути об’єкта цього класу. Екземпляр успадковує значення атрибутів класу. Для перегляду атрибутів застосовуйте атрибут `__dict__`, який містить словник з атрибутами об’єкта, і функцію `dir`, яка повертає список атрибутів об’єкта.

```
class A(): # клас A
    a=0 # атрибут класу
class B(A): # клас B
    b=0 # атрибут класу
A.a=1 # зміна значення атрибута класу
A.x=2 # створення нового атрибута класу та зміна його
значення
print A.__dict__ # {'a': 1, 'x': 2, '__module__':
'__main__', '__doc__': None}
obj=A() # створити екземпляр класу A
print obj.a # 1
print obj.x # 2
A.a=3 # зміна значення атрибута класу
obj.a=4 # зміна значення атрибута екземпляра
print A.__dict__ # {'a': 3, 'x': 2, '__module__':
'__main__', '__doc__': None}
print obj.__dict__ # {'a': 4}
print B.__dict__ # {'__module__': '__main__', 'b': 0,
'__doc__': None}
print dir(B) # ['__doc__', '__module__', 'a', 'b',
'x']
print B.a # 3
B.a=5 # зміна значення атрибута класу
print B.a # 5
print A.a # 3
```

```
{'a': 1, 'x': 2, '__module__': '__main__', '__doc__':
None}
1
2
{'a': 3, 'x': 2, '__module__': '__main__', '__doc__':
None}
{'a': 4}
{'__module__': '__main__', 'b': 0, '__doc__': None}
['__doc__', '__module__', 'a', 'b', 'x']
```

```
3
5
3
```

Статичні методи та методи класу

Статичний метод - це функція, яка визначена в класі, але не належить класу чи екземпляру. Метод класу - це метод, який належить класу, а не екземпляру. Метод класу має перший аргумент `cls` (клас), а не `self` (екземпляр). Статичні методи і методи класу визначаються за допомогою декораторів `@staticmethod` і `@classmethod`.

```
class A: # визначення класу A
    a=0 # атрибут класу
    def f(self, x): # метод екземпляра
        return self.a+x
    @staticmethod # декоратор
    def f2(x): # статичний метод
        return 1+x
    @classmethod # декоратор
    def f3(cls, x): # метод класу
        return cls.a+x
obj=A() # створити об'єкт (екземпляр)
obj.a=2 # атрибут екземпляра (не класу!)
print obj.f(1) # виклик методу екземпляра
print A.f(obj, 1) # або
print A.f2(1) # виклик статичного методу
print A.f3(1) # виклик методу класу
```

```
3
3
2
1
```

Властивості

Властивість - це атрибут, який володіє методами читання, запису і знищення значення. Під час присвоювання властивості значення викликається метод запису, а під час отримання значення властивості - метод читання. Властивості можна створювати в класах, які успадковані від `object`, за допомогою функції `property` або за допомогою декоратора `@property`.

```
class A(object): # клас A успадкований від object
    __x=0 # приватний атрибут __x
    def getx(self): return self.__x # метод читання
    def setx(self, x): # метод запису
        if x>0: # якщо x>0
            self.__x = x # присвоїти x
        else: self.__x=0 # інакше присвоїти 0
    x = property(getx, setx, None, "Property x") # 
властивість x з методами читання, запису і рядком 
документації
a=A() # створити об'єкт класу A
a.x=-2 # присвоїти властивості x значення
print a.x # вивести значення властивості x
```

```
0
```

Перевантаження операторів

Перевантаження (перевизначення) операторів - це можливість зміни функціонування стандартних операторів (`+` , `-` , `==`, `()` та ін.) для об'єктів користувача. Для цього в класах цих об'єктів створюються реалізації відповідних методів (`__add__`, `__sub__`, `__eq__`, `__call__` та ін.)

```
class A(object): # клас A
    x=0 # атрибут-дане x
    def __init__(self,x=0): # конструктор
        self.x=x
```

```
    def __add__(self, obj): # метод перевантажує
оператор +
        return A(self.x+obj.x) # повертає об'єкт
класу A
    def __eq__(self, obj): # метод перевантажує
оператор ==
        return self.x==obj.x # повертає значення
логічного виразу
    def __call__(self,x=0): # метод перевантажує
оператор ()
        return A(x) # повертає об'єкт класу A
    def __str__(self): # метод повертає рядкове
відображення об'єкта
        return "%s"%self.x
a=A(2); b=A(2) # створити об'єкти a і b класу A
c=a+b # виклик перевантаженого оператора +
print type(c) # тип об'єкта c
print a==b # виклик перевантаженого оператора ==
print a(3)+b(2) # виклик перевантажених операторів ()
і +
print c # виклик методу __str__
```

```
<class '__main__.A'>
True
5
4
```

Контейнери

Контейнер - це структура даних, яка зберігає інші об'єкти в організованому вигляді. Як правило клас контейнера містить методи `__iter__`, `next`, `__getitem__`. Приклад показує створення класу контейнера `Container` і його використання. Див. також модуль `collections`.

```
class Container(object): # клас контейнера, успадкований від object
    def __init__(self,lst): # конструктор
        self.lst=lst # атрибут-дане список
        self.current=-1 # поточний індекс
    def __iter__(self): # метод повертає ітератор
        return self
    def next(self): # повертає наступний елемент контейнера
        # якщо індекс наступного елемента менший довжини контейнера
        if self.current+1<len(self.lst):
            self.current=self.current+1 # збільшити індекс поточного
            return self.lst[self.current] # повернути поточний
        else: # інакше
            raise StopIteration # генерувати StopIteration
    def __getitem__(self,i): # метод повертає елемент за індексом `i`
        return self.lst[i]
c=Container([1,2,3,4,5]) # створити об'єкт контейнера
for i in c: # для кожного елемента в контейнері
    print i, # вивести його
print
print c[0] # вивести перший елемент (або c.lst[0])
it=iter(c) # створити об'єкт ітератор (або так: c.__iter__())
c.current=0 # установити поточний індекс
print it.next(),it.next(),c[3] # вивести два наступні та четвертий
```

```
1 2 3 4 5
1
2 3 4
```

Менеджери контексту і інструкція with

Менеджер контексту - це об’єкт, який визначає контекст (середовище) виконання інструкцій всередині блоку `with`. Містить методи `__enter__` та `__exit__`, які автоматично викликаються на початку і вкінці блоку `with`. Часто використовується під час роботи з файлами. Приклад показує створення і використання класу менеджера контексту.

```
with open(__file__,'r') as f: # закриє файл автоматично
    print f.read(1)

class A(object): # клас реалізує власний спосіб керування контекстом
    def __init__(self,a):
        self.a=a
    def __enter__(self): # викликається під час входу в `with`
        print 'with enter'
        return self # obj=self
    def __exit__(self,type,value,tb): # викликається під час виходу з `with`
        print 'with exit'
        return False
with A(1) as obj: # тут викликається __enter__
    print obj.a
# тут викликається __exit__
```

```
#
with enter
```

```
1
with exit
```

Метакласи

Метакласи - це об'єкти, які створюють класи. Відомим метакласом є функція `type`. Метакласи використовуються для створення класів на етапі виконання. Нижче показані різні способи використання метакласів.

```
def cls_factory(a,fn): # функція створює новий клас з атрибутами `a`,`fn`
    class C(object):pass # пустий клас, успадкований від object
    setattr(C,'a',a) # установити атрибут `a`
    setattr(C, fn.__name__, fn) # установити атрибут `fn`
    return C # повернути клас
def method1(self): # метод класу
  print self.a # вивести значення атрибута `a`
Class1 = cls_factory(1,method1) # створити клас Class1
obj1 = Class1() # створити об'єкт obj1
obj1.method1() # викликати метод method1

#створити клас за допомогою метакласу type
Class2 = type('Class2', (object,), {'a':2,'method1': method1})
obj2=Class2() # створити об'єкт obj2
obj2.method1() # викликати метод method1

class My_Type(type): # створити метаклас, який успадковує type
    def __new__(cls, name, bases, dict): # метод створення класу
        return type.__new__(cls, name, bases, dict) #
```

```
виклик __new__ базового класу
    def __init__(cls, name, bases, dict): # метод
ініціалізації класу
        return type.__init__(cls, name, bases, dict)
# виклик __init__ базового класу
# створити клас за допомогою метакласу
Class3 = My_Type('Class3', (object,),
{'a':3,'method1': method1})
obj3=Class3() # створити об'єкт obj3
obj3.method1() # викликати метод method1
```

```
1
2
3
```

Декоратори

Декоратор - це функція-обгортка, яка отримує і повертає іншу функцію, метод чи клас. Використовуються для розширення їх можливостей.

```
def decorator(fn): # функція-обгортка, яка отримує і
повертає fn
    print 'y=', # додані нові можливості
    return fn # повертає функцію fn
def function(x): # функція, яка обгортається
    return x*x
function=decorator(function) # обгорнути функцію
print function(2) # виклик обгорнутої функції

# те саме, але з застосуванням декоратора @decorator
@decorator
def function(x): # функція, яка обгортається
    return x*x
print function(2) # виклик обгорнутої функції
```

```
y= 4
y= 4
```

Декоратори з аргументом

Декоратор може мати довільні аргументи. В прикладі декоратор має аргумент `arg`, значення якого виводиться перед викликом функції, що обгортається.

```
def decorator(arg): # функція отримує аргумент і повертає внутрішню функцію f
    def f(fn): # внутрішня функція-обгортка
        print arg, # додані нові можливості
        return fn # повертає функцію fn
    return f
def function(x): # функція, яка обгортається
    return x*x
temp=decorator('y=') # отримати аргумент і повернути функцію f
function=temp(function) # обгорнути функцію
print function(2) # виклик обгорнутої функції

# те саме, але з застосуванням декоратора @decorator з аргументом
@decorator('y=')
def function(x): # функція, яка обгортається
    return x*x
print function(2) # виклик обгорнутої функції
```

```
y= 4
y= 4
```

Декоратори класу

За тим самим принципом можна обгортати класи. Приклад показує як за допомогою декоратора класу автоматично змінювати значення його атрибута `__name__`.

```
def decorator(arg): # функція отримує аргумент і повертає внутрішню функцію f
    def f(cls): # внутрішня функція-обгортка отримує і повертає клас
        cls.__name__=arg # змінити значення атрибута класу
        return cls # повертає клас
    return f
# застосування декоратора класу з аргументом
@decorator('Мій клас')
class A(object): # клас
    a=1
print A.__name__
```

```
Мій клас
```

Інтроспекція

Інтроспекція в Python - це можливість отримати всю інформацію про структуру будь-якого об’єкта під час виконання. Найбільш відомим засобом для інтроспекції в Python є функція `dir`, яка повертає список імен атрибутів переданого їй об’єкта. Функція `type` або атрибут `__class__` дозволяють отримати тип об’єкта. Функція `vars` або атрибут `__dict__` дозволяють отримати словник з парами атрибут:значення об’єкта. Функції `hasattr`, `getattr` і `setattr` дозволяють відповідно перевірити наявність у об’єкта заданого атрибута, повернути його і змінити значення. Функція `issubclass` дає змогу визначити чи успадковується один клас від іншого, а метод `__subclasses__` повертає список підкласів. Кортеж базових класів та їх ієрархію можна отримати за допомогою атрибутів `__bases__` і `__mro__`.

```
class A(object): # успадкований від object клас A
    '''Клас A''' # рядок документації
    def __init__(self,a): # конструктор
        self.a=a # атрибут a
```

```
    def f(self): # метод f
        '''Повертає self.a''' # рядок документації
        return self.a
class B(A): # успадкований від A клас B
    '''Клас B''' # рядок документації
    def __init__(self,a,b): # конструктор
        super(B, self).__init__(a) # виклик
конструктора базового класу
        self.b=b # атрибут b
    def f(self): # метод f
        '''Повертає суму self.a+self.b''' # рядок
документації
        return self.a+self.b

obj=B(0,2) # створення об'єкта класу B, виклик
конструктора
obj.a=1 # зміна значення атрибута a
obj.f() # виклик методу f

print dir(B) # список імен атрибутів класу B
print dir(obj) # список імен атрибутів об'єкта obj
print id(obj) # унікальний ідентифікатор об'єкта
print obj.__sizeof__() # розмір об'єкта в пам'яті в
байтах
print B.__doc__ # рядок документації класу B
print obj.f.__doc__ # рядок документації методу f
print B.__name__ #  ім'я класу B
print __name__ # ім'я модуля
print type(obj) # тип (клас) об'єкта obj
# або obj.__class__
print obj.__class__.__name__ # ім'я типу об'єкта obj
print vars(obj) # словник з парами атрибут:значення
# або obj.__dict__
print hasattr(obj, 'a') # чи є атрибут 'a' у об'єкта
obj?
```

```
setattr(obj, 'a', 3) # зміна значення (3) атрибута
'a' об'єкта obj
# або obj.__setattr__('a',3)
print getattr(obj, 'a') # значення атрибута 'a'
об'єкта obj
# або obj.__getattribute__('a')
# або obj.__dict__['a']
print callable(obj.f) # чи атрибут f є методом?
print isinstance(obj, B) # чи obj є екземпляром B?
print issubclass(A, object) # чи A є підкласом
object?
print A.__subclasses__() # підкласи A
print B.__bases__ # кортеж базових класів
print B.__mro__ # кортеж з ієрархією базових класів
```

```
['__class__', '__delattr__', '__dict__', '__doc__', '
__format__', '__getattribute__', '__hash__', '__init_
_', '__module__', '__new__', '__reduce__', '__reduce_
ex__', '__repr__', '__setattr__', '__sizeof__', '__st
r__', '__subclasshook__', '__weakref__', 'f']
['__class__', '__delattr__', '__dict__', '__doc__', '
__format__', '__getattribute__', '__hash__', '__init_
_', '__module__', '__new__', '__reduce__', '__reduce_
ex__', '__repr__', '__setattr__', '__sizeof__', '__st
r__', '__subclasshook__', '__weakref__', 'a', 'b', 'f
']
101226368
32
Клас B
Повертає суму self.a+self.b
B
__main__
<class '__main__.B'>
B
{'a': 1, 'b': 2}
```

```
True
3
True
True
True
[<class '__main__.B'>]
(<class '__main__.A'>,)
(<class '__main__.B'>, <class '__main__.A'>, <type 'o
bject'>)
```

inspect - перегляд об’єктів часу виконання

Модуль inspect містить додаткові функції, які допомагають отримати інформацію про об’єкти часу виконання (модулі, класи, методи, функції, об’єкти трасування, кадрів виконання і коду).

```
import inspect
# клас A
class A(): pass
print inspect.getmro(A) # кортеж з ієрархією базових
класів
print inspect.getmembers(A) # повертає список пар
(ім'я, значення) членів об'єкта
print inspect.getcomments(A) # коментар перед класом
A
#print inspect.getsource(A) # текст вихідного коду
класу A
print inspect.isclass(A) # чи A є класом?

def f(a,b=0,*args,**kwargs):
    cf=inspect.currentframe() # об'єкт поточного
кадру виконання
    #cf=sys._getframe() # або
    #cf.f_back # попередній кадр стеку (який викликав
f)
    print cf.f_lineno, cf.f_back.f_lineno # поточний
```

```
рядок коду і рядок, який викликав f
    print cf.f_locals # локальні імена f
    #print cf.f_back.f_code.co_filename # файл
модуля, що викликав f

print inspect.ismethod(f) # чи f є методом?
print inspect.isfunction(f) # чи f є функцією?
print inspect.getargspec(f) # імена аргументів
функції
f(1,2,3,x=4) # виклик функції
```

```
(<class __main__.A at 0x0000000005F61D68>,)
[('__doc__', None), ('__module__', '__main__')]
None
True
False
True
ArgSpec(args=['a', 'b'], varargs='args', keywords='kw
args', defaults=(0,))
14 21
{'a': 1, 'args': (3,), 'b': 2, 'cf': <frame object at
0x0000000005F50930>, 'kwargs': {'x': 4}}
```

copy - копії об’єктів

Модуль copy призначений для створення поверхневих і глибоких копій складених об’єктів. Функція copy створює поверхневу копію шляхом копіювання посилань на атрибути об’єкта. Функція deepcopy створює глибоку копію шляхом рекурсивного створення окремих копій атрибутів об’єкта.

```
import copy
class A(object): pass # клас A
class B(object): pass # клас B
a=A() # об'єкт a
b=B() # об'єкт b
```

```
a.x=b # атрибут x
b.y=5 # атрибут y
copy_a=copy.copy(a) # поверхнева копія об'єкта
print a, a.x, a.x.y
print copy_a, copy_a.x, copy_a.x.y
copy_a=copy.deepcopy(a) # повністю незалежна глибока
копія об'єкта
print copy_a, copy_a.x, copy_a.x.y
```

```
<__main__.A object at 0x03653490> <__main__.B object
at 0x03653810> 5
<__main__.A object at 0x03653AD0> <__main__.B object
at 0x03653810> 5
<__main__.A object at 0x03653A70> <__main__.B object
at 0x03653B10> 5
```

itertools - функції для ефективних ітерацій

Модуль itertools містить функції, які створюють ітератори і призначені для ефективних ітерацій по даним. Для економії пам'яті ітератори застосовуються з оператором `for`, але в прикладі вони передані функції `list`. Це зроблено тільки для зменшення об'єму коду прикладу.

```
from operator import add # бінарний оператор +
from itertools import *
print list(izip('ab', 'cd')) # зшиває перший з
першим, другий з другим  і т.д.
# count - послідовні значення, cycle - повторює
послідовність нескінченно
for n,i in izip(count(), cycle('abc')):
    if n>5: break
    print (n,i),
print
print list(chain('ab','cd')) # об'єднує в один
print list(compress('abcd', [1,0,1,0])) # тільки ті
```

```
елементи, яким відповідає 1
print list(dropwhile(lambda x: x!='c', 'abcd')) #
відкидати поки True
print list(takewhile(lambda x: x!='c', 'abcd')) #
приймати поки True
print [(k,list(g)) for k, g in groupby('aaabbac')] #
групує
print list(ifilter(lambda x: x in 'bd', 'abcd')) #
фільтрує
print list(imap(add, (1,2,3), (4,5,6))) # add(1,4),
add(2,5), ...
#print list(imap(lambda x,y: x+y, (1,2,3), (4,5,6)))
# або
print list(starmap(add, [(1,2), (4,5)])) # add(1,2),
add(4,5), ...
print [list(i) for i in tee('abc',3)] # створює 3
незалежні ітератори
print "Комбінаторні генератори:"
print list(product('ab','cd')) # декартів добуток
print list(product('ab',repeat=2)) # декартів добуток
з собою
#print list(product('ab','ab')) # або
print list(permutations('abc',2)) # усі можливі
перестановки з двох елементів
print list(combinations('abc',2)) # усі можливі
комбінації з двох елементів
print list(combinations_with_replacement('abc',2)) #
тут дозволені повтори
```

```
[('a', 'c'), ('b', 'd')]
(0, 'a') (1, 'b') (2, 'c') (3, 'a') (4, 'b') (5, 'c')
['a', 'b', 'c', 'd']
['a', 'c']
['c', 'd']
['a', 'b']
```

```
[('a', ['a', 'a', 'a']), ('b', ['b', 'b']), ('a', ['a
']), ('c', ['c'])]
['b', 'd']
[5, 7, 9]
[3, 9]
[['a', 'b', 'c'], ['a', 'b', 'c'], ['a', 'b', 'c']]
Комбінаторні генератори:
[('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd')]
[('a', 'a'), ('a', 'b'), ('b', 'a'), ('b', 'b')]
[('a', 'b'), ('a', 'c'), ('b', 'a'), ('b', 'c'), ('c'
, 'a'), ('c', 'b')]
[('a', 'b'), ('a', 'c'), ('b', 'c')]
[('a', 'a'), ('a', 'b'), ('a', 'c'), ('b', 'b'), ('b'
, 'c'), ('c', 'c')]
```

re - операції з використанням регулярних виразів

Модуль re забезпечує операції з використанням регулярних виразів. Регулярний вираз (РВ) - це послідовність символів (шаблон), яка відповідає певній множині рядків [28]. Зазвичай використовуються для операцій пошуку чи заміни рядків. Наприклад, шаблону ‘.o’ в рядку ‘Hello World’ відповідають рядки ‘lo’ та ‘Wo’. РВ може містити звичайні (як ‘o’) і спеціальні (як ‘.’) символи. Для прикладу, спеціальний символ ‘.’ означає будь-який символ окрім символу нового рядка. Спеціальні символи сприймаються як звичайні, якщо перед ними стоїть символ ‘\’. Шаблони і рядки для пошуку можуть бути 8-бітними рядками або Юнікод-радками. Створення РВ можна суттєво спростити за допомогою таких програм як Kodos, RegexBuddy або regex101.com.

```
from __future__ import print_function
import re
s='Hello World' # рядок для операцій

mo=re.search('World', s) # знаходить у s першу
відповідність шаблону
```

```
print(mo.group(0))
#World

mo=re.match('Hello', s) # знаходить на початку s
першу відповідність шаблону
print(mo.group(0))
#Hello

po=re.compile('o') # компілює шаблон в об'єкт
регулярного виразу
mo=po.search(s) # знаходить у s першу відповідність
шаблону
print(mo.group(0), mo.span()) # вміст знайденого
(групи), початок і кінець
#o (4, 5)
# функції об'єктів регулярного виразу мають параметри
pos і endpos:
mo=po.search(s,pos=7,endpos=10) # знаходить у s першу
відповідність шаблону (шукає з 7 по 10)
print(mo.group(0), mo.span()) # вміст знайденого
(групи), початок і кінець
#o (7, 8)

mo=re.search('(H).*(W)', s) # пошук за шаблоном з
групами
print(mo.groups()) # усі групи
#('H', 'W')
print(mo.group(0)) # група 0 (рядок, що відповідає
повному шаблону)
#Hello W
print(mo.group(1)) # група 1 (рядок, що відповідає H)
#H
print(mo.group(2)) # група 2 (рядок, що відповідає W)
#W
print(mo.group(1,2))
```

```
#('H', 'W')
print(mo.start(),mo.end()) # початок і кінець групи 0
#0 7
print(mo.start(2),mo.end(2)) # початок і кінець групи
2
#6 7
print(mo.span(2)) # або
#(6, 7)
print(mo.expand(r'\1ello \2orld')) # підставляє вміст
груп 1 і 2
#Hello World

mo=re.search('(?P<name1>H).*(?P<name2>W)', s) # пошук
за шаблоном з іменованими групами
print(mo.groupdict()) # словник груп
#{'name2': 'W', 'name1': 'H'}

print(re.findall('o', s)) # усі відповідності, що не
перекриваються
#['o', 'o']

for mo in re.finditer('o',s): # те саме, але ітератор
    print(mo.group(0))
#o
#o

print(re.split(' ', s)) # розділює за шаблоном
#['Hello', 'World']

print(re.sub(' ','_',s)) # заміна за шаблоном
#Hello_World
print(re.subn(' ','_',s)) # або показувати кількість
зроблених замін
#('Hello_World', 1)
```

```
print(re.sub(r'"(.*?)"',r'<a href="\g<1>">\g<1></a>',
r'"dir\file.html"')) # заміна з використанням груп
(\g<1>)
#<a href="dir\file.html">dir\file.html</a>

def repl(mo): # повертає новий рядок, яким замінює
re.sub
    path=mo.group(1) # рядок знайденої групи
    return "["+path+"]"
pattern=re.compile(u'<img src="(.*?)" />') # що
заміняти
print(re.sub(pattern, repl, u'***<img src="1.png"
/>***')) # замінити все
#***[1.png]***

print(re.escape(s)) # екранує не алфавітно-цифрові
символи
#Hello\ World

print(re.findall('.', 'Hello')) # будь-який символ
#['H', 'e', 'l', 'l', 'o']
print(re.findall('^.', 'Hel\nlo')) # символ на
початку рядка
#['H']
print(re.findall('^.', 'Hel\nlo',re.MULTILINE))
#['H', 'l']
print(re.findall('.$', 'Hel\nlo')) # символ вкінці
рядка
#['o']
print(re.findall('.$', 'Hel\nlo',re.MULTILINE))
#['l', 'o']
print(re.findall('L', 'HELLO')) # символ L
#['L', 'L']

print(re.findall('L*', 'HELLO')) # 0 і більше L
```

```
#['', '', 'LL', '', '']
print(re.findall('L+', 'HELLO')) # 1 і більше L
#['LL']
print(re.findall('LL?', 'HELLO')) # 0 або 1 L
#['LL']
print(re.findall('L{2}', 'HELLO')) # 2 L
#['LL']
print(re.findall('L{2,5}', 'HELLO')) # від 2 до 5 L
#['LL']

# те саме, але шукають і поглинають мінімальну
кількість символів:
print(re.findall('L*?', 'HELLO'))
#['', '', '', '', '', '']
print(re.findall('L+?', 'HELLO'))
#['L', 'L']
print(re.findall('LL??', 'HELLO'))
#['L', 'L']
print(re.findall('L{2}?', 'HELLO'))
#['LL']
print(re.findall('L{2,5}?', 'HELLO'))
#['LL']

print(re.findall('[EO]', 'HELLO')) # символи E або O
#['E', 'O']
print(re.findall('[a-zA-Z0-9]', 'HELLO')) # усі букви
і цифри
#['H', 'E', 'L', 'L', 'O']
print(re.findall('[^EO]', 'HELLO')) # не символи E
або O
#['H', 'L', 'L']

print(re.findall('\*\?\+\|\(\)', '*?+|()')) #
екранування спеціальних символів
#['*?+|()']
```

```
print(re.findall(r'\\', r''+'\\'))
#['\\']

print(re.search(r'(E).*(O)\1', 'HELLOE').group(0)) # \1 - вміст першої групи
#ELLOE
print(re.search(r'(?P<name>E).*(O)(?P=name)', 'HELLOE').group(0)) # або (?P=name) - вміст групи (?P<name>E)
#ELLOE

print(re.findall('E|O', 'HELLO')) # знайти E або O
#['E', 'O']
print(re.findall('EO', 'HELLO')) # знайти EO
#[]

print(re.search('(E)', 'HELLO').group(1)) # вміст першої групи (E)
#E
print(re.search('(?P<name>E)', 'HELLO').group(1)) # вміст групи (?P<name>E)
#E
print(re.search('(?P<name>E)', 'HELLO').group('name')) # або
#E
print(re.search('(?:E)', 'HELLO').group(0)) # не створює групу
#E

print(re.findall('E(?=L)', 'HELLO')) # якщо наступний символ L
#['E']
print(re.findall('E(?!L)', 'HELLO')) # якщо наступний символ не L
#[]
```

```
print(re.findall('(?<=L)E', 'HELLO')) # якщо попередній символ L
#[]
print(re.findall('(?<!L)E', 'HELLO')) # якщо попередній символ не L
#['E']

print(re.findall('E(?#comment)', 'HELLO')) # коментар (?#comment)
#['E']
print(re.search(r'(<)(\d*)(?(1)>)', 'xx<12>xx').group(2)) # якщо група 1 містить <, то шукати >
#12

# флагі режиму:
print(re.findall('(?s).', 'HEL\nLO')) # враховувати символ \n
#['H', 'E', 'L', '\n', 'L', 'O']
print(re.findall('.', 'HEL\nLO')) # те саме без (?s)
#['H', 'E', 'L', 'L', 'O']
print(re.findall('(?i)E', 'HeLLO')) # не чутливий до регістру
#['e']
print(re.findall(u'E', u'HeLLO', re.IGNORECASE | re.UNICODE)) # або так для Unicode
#[u'e']
print(re.findall('(?x)   E    ', 'HELLO')) # не чутливий до пробілів
#['E']

# спеціальні послідовності:
print(re.findall(r'\A', 'HELLO')) # початок рядка
#['']
print(re.findall(r'\Z', 'HELLO')) # кінець рядка
```

```
#['']
print(re.findall(r'HEL\b', 'HEL\nLO')) # пустий рядок
на границі слова
#['HEL']
print(re.findall(r'HEL\B', 'HEL\nLO')) # пустий рядок
не на границі слова
#[]
print(re.findall(r'\d', '123')) # будь-яка десяткова
цифра
#['1', '2', '3']
print(re.findall(r'\D', '123')) # не цифра
#[]
print(re.findall(r'\s', ' \t\n\r\f\v')) # будь-який
пробільний символ
#[' ', '\t', '\n', '\r', '\x0c', '\x0b']
print(re.findall(r'\S', ' \t\n\r\f\v')) # будь-який
не пробільний символ
#[]
print(re.findall(r'\w', 'HELLO')) # будь-який
алфавітно-цифровий символ
#['H', 'E', 'L', 'L', 'O']
print(re.findall(r'\W', 'HELLO')) # будь-який не
алфавітно-цифровий символ
#[]
```

decimal - дійсні числа довільної точності

На відміну від типу даних `float`, модуль `decimal` дозволяє точно подавати дробові десяткові значення.

```
import sys
import decimal # модуль для арифметики довільної
точності
print 0.1*7==0.7 # False
print decimal.Decimal('0.1')*7 ==
decimal.Decimal('0.7') # True
```

```
print sys.float_info # інформація про тип float
x=1.7976931348623157e+308 # найбільше float
print 2*x # результат: inf
x=decimal.Decimal('1.7976931348623157e+308') # дійсне довільної точності
print x.as_tuple() # кортеж у вигляді (знак, мантиса, порядок)
print 2*x # результат: 3.595386269724631E+308
```

```
False
True
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
inf
DecimalTuple(sign=0, digits=(1, 7, 9, 7, 6, 9, 3, 1, 3, 4, 8, 6, 2, 3, 1, 5, 7), exponent=292)
3.595386269724631E+308
```

time - визначення і конвертування значень часу

Функції для визначення і конвертування значень часу. Дивись також модулі `datetime` і `calendar`.

```
import time
print time.clock() # час CPU в секундах з часу першого запуску цієї функції
time.sleep(1.0) # зупинити виконання на 1 секунду
print time.clock() # буде приблизно 1
print time.time() # кількість секунд з початку Епохи (1.1.1970 р.)
print time.localtime() # поточний час (struct_time)
print time.localtime()[0] # поточний рік
print '*',time.localtime(1424196030) # заданий час (struct_time)
```

```
print time.timezone # зона часу як зміщення в
секундах
```

```
1.6364630143e-06
1.00987850899
1535381596.49
time.struct_time(tm_year=2018, tm_mon=8, tm_mday=27,
tm_hour=17, tm_min=53, tm_sec=16, tm_wday=0, tm_yday=
239, tm_isdst=1)
2018
* time.struct_time(tm_year=2015, tm_mon=2, tm_mday=17
, tm_hour=20, tm_min=0, tm_sec=30, tm_wday=1, tm_yday
=48, tm_isdst=0)
-7200
```

datetime - робота з датою і часом

Класи для роботи з датою і часом. Містить класи `date` (дата), `time` (час), `datetime` (дата і час), `timedelta` (період часу), `tzinfo` (абстрактний клас для роботи з часовими поясами). Дозволяє виконувати різноманітні математичні операції над значеннями дати і часу.

```
import datetime, time
d0=datetime.datetime.now() # поточна дата (datetime)
d1=datetime.datetime(2015,2,17,0,0,0,0) # задана дата
(datetime)
print d0.year, d0.month, d0.day, d0.hour, d0.minute,
d0.second, d0.microsecond, d0.tzinfo # атрибути
datetime
st=d0.timetuple() # об'єкт (time.struct_time)
print time.mktime(st) # кількість секунд з початку
Епохи
print datetime.datetime.strptime("Mon Apr 26 11:31:53
2010", "%a %b %d %H:%M:%S %Y").strftime("%d%M%Y") #
конвертація з формату в формат
```

```
print d0.isoweekday() # ISO день тиждня (1 -
понеділок)
print d1.replace(year=2014) # дозволяє змінити певні
атрибути дати

# тут місяців і років немає, бо вони можуть мати
різну к-сть днів
td1=datetime.timedelta(weeks=1,days=2,hours=1)
print td1.days, td1.seconds, td1.microseconds
print d1+td1 # додати до дати період (datetime)
print d1+td1*2-abs(-td1) # допустимі операції
(datetime)
td2=d0-d1 # різниця дат (datetime)
print d0>d1 # порівняння дат
print td1>td2 # порівняння періодів
# datetime знає про високосні роки:
print datetime.datetime(2016,3,1)-
datetime.datetime(2016,2,28) # результат 2 дня
#print datetime.datetime(2015,2,29) # помилка!
```

```
2018 8 27 18 5 13 543000 None
1535382313.0
26312010
1
2014-02-17 00:00:00
9 3600 0
2015-02-26 01:00:00
2015-02-26 01:00:00
True
False
2 days, 0:00:00
```

calendar - робота з календарем

Функції для виведення календаря і роботи з ним. За замовчуванням першим днем тиждня є понеділок, а останнім - неділя.

```
import calendar
c=calendar.Calendar(calendar.MONDAY) # календар (або calendar.Calendar())
print [d for d in c.itermonthdates(2016, 2)][:2] # ітератор на дні місяця datetime.date (цілі тиждні)
print calendar.weekday(2016, 2, 29) # день тиждня
print calendar.monthrange(2016, 2) # день тиждня першого дня місяця і кількість днів в місяці
calendar.TextCalendar(calendar.MONDAY).formatmonth(2016, 2) # повертає текстовий календар на місяць
calendar.LocaleTextCalendar(calendar.MONDAY,'Ukrainian_Ukraine').formatmonth(2016, 2) # повертає текстовий календар на місяць (українська мова)
calendar.HTMLCalendar(calendar.MONDAY).formatmonth(2016, 2) # повертає html календар на місяць
```

```
[datetime.date(2016, 2, 1), datetime.date(2016, 2, 2)
]
0
(0, 29)
```

pdb - відлагоджувач Python

pdb - інтерактивний відлагоджувач (debugger) вихідного коду Python-програм, який дозволяє установлення точок зупинки, виконання в покроковому режимі, обчислення довільних Python-виразів, перегляд кадрів стеку та поставарійне відлагодження. Виконайте програму так:

```
python main.py
```

І введіть послідовно наступні команди відлагожувача:

`n` (виконувати до наступного рядка)
`s` (виконати рядок і зупинитись в функції, що викликається)
`a` (вивести аргументи функції)
`p x` (вивести значення `x`)
`!x` (або так)
`!x=2` (змінити значення `x`)
`r` (виконувати до виходу з функції)
`c` (продовжити до точки зупинки)

Відлагоджувач можна також викликати командою:

```
python -m pdb main.py
```

Щоб вийти з відлагоджувача введіть команду `q`. Для ознайомлення з поставарійним відлагодженням закоментуйте рядок `import pdb; pdb.set_trace()` і введіть в консолі команди:

```
python
>>> import main      # тут виникне ZeroDivisionError
>>> import pdb; pdb.pm()
x
q
>>> exit()
```

```
def f(x):
    return 1.0/x
import pdb; pdb.set_trace() # точка зупинки (ввійти в відлагоджувач)
print f(1)
print f(0)
```

timeit - тривалість виконання невеликих частин коду

Модуль `timeit` дозволяє просто визначати тривалість виконання невеликих частин коду. Для великих частин коду

використовуйте модуль `time`. З прикладу видно, що `sin(x)` виконується швидше ніж `math.sin(x)`.

```
import timeit
print timeit.timeit('math.sin(x)', setup='import math; x = 1', number=1000000) # час  виконання 1000000 раз (секунд)
print timeit.timeit('sin(x)', setup='from math import sin; x = 1', number=1000000)
```

```
0.262688315981
0.192319588135
```

logging - ведення журналу

В цьому модулі визначені функції і класи, які реалізують гнучку систему реєстрації подій для прикладних програм і бібліотек. Нижче показано найпростіший спосіб використання модуля. Приклад створює файл `mylog.log` з журналом подій.

```
import logging
logging.basicConfig(format='%(levelname)-8s [%(asctime)s]  %(message)s', level=logging.DEBUG, filename='mylog.log', filemode='w') # конфігурування системи реєстрації подій
logging.debug('Повідомлення налагоджувача')
logging.info('Інформаційне повідомлення')
logging.warning('Попередження')
logging.error('Помилка')
logging.critical('Критичне повідомлення')
```

```
DEBUG    [2018-08-31 14:56:40,039]  Повідомлення нала
годжувача
INFO     [2018-08-31 14:56:40,039]  Інформаційне пові
домлення
WARNING  [2018-08-31 14:56:40,039]  Попередження
```

```
ERROR    [2018-08-31 14:56:40,039]  Помилка
CRITICAL [2018-08-31 14:56:40,039]  Критичне повідомл
ення
```

pickle - серіалізація об'єктів Python

Серіалізація - це процес перетворення якої-небудь структури даних в послідовність бітів. Часто використовується для передачі об'єктів по мережі або для збереження їх у файли. Модуль pickle реалізує алгоритми для серіалізації і десеріалізації об'єктів Python.

```
import pickle
with open('data.pkl', 'wb') as f: # відкрити бінарний 
файл для запису
    pickle.dump([1,2,3], f) # серіалізувати список у 
файл
    pickle.dump([4,5,6], f)
with open('data.pkl', 'rb') as f: # відкрити бінарний 
файл для читання
    print pickle.load(f) # десеріалізувати список
    print pickle.load(f)
s=pickle.dumps([7,8,9]) # серіалізувати список в 
рядок
print pickle.loads(s) # десеріалізувати список
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

shelve - збереження об'єктів Python

Модуль shelve (полиця) призначений для збереження у постійній пам'яті Python-об'єктів (які може зберігати pickle) в об'єкті, подібному на словник.

```
import shelve
d=shelve.open("file.dat") # відкрити файл полиці
```

```
d["1"]=[1,2,3] # записати у полицю об'єкт під ключем
"1"
d["2"]=[4,5,6]
d.close() # закрити файл полиці

d = shelve.open("file.dat")
if d.has_key("2"): # якщо є ключ "2"
    del d["2"] # видалити об'єкт під ключем "2"
d.sync() # зберегти усі зміни на диску
print d.keys() # вивести список усіх ключів
if d.has_key("1"): print d["1"] # об'єкт під ключем
"1"
d.close()
```

```
['1']
[1, 2, 3]
```

anydbm - універсальний доступ до DBM баз даних

Модуль anydbm реалізує універсальний доступ до різних DBM баз даних (БД). Для доступу до бази даних використовується подібний на словник інтерфейс. На відміну від shelve, ключі і значення словника повинні бути рядками.

```
import anydbm
db=anydbm.open("mydbm.db", "c") # відкрити БД для
читання і запису, створити, якщо не існує
db["Іванов"]="1990" # записати у БД (зверніть увагу -
рядки!)
db["Петров"]="1992"
for k,v in db.iteritems(): # вивести усі записи бази
даних
    print k,v # ключ і значення
db.close() # закрити БД
```

```
Петров 1992
Іванов 1990
```

sqlite3 - DB-API 2.0 інтерфейс для баз даних SQLite

SQLite – це бібліотека, яка реалізує систему керування реляційними базами даних. Підтримує транзакції, не потребує інсталяції, створена мовою C, швидка, не залежить від платформи. Взаємодія з базою даних відбувається мовою SQL. Модуль sqlite3 є інтерфейсом Python до SQLite.

```
import sqlite3
conn = sqlite3.connect('mysqlite3.db') # об'єкт бази
даних
cur = conn.cursor() # об'єкт курсор - виконує запити
і отримує результати
# виконати команду SQL, яка створює таблицю з полями
name і content
cur.execute('CREATE TABLE IF NOT EXISTS pages(name
TEXT, content TEXT)')
conn.commit() # зберегти зміни
# база даних блокується поки транзакція не
виконається

name=u"Іванов"
content=u"1990"
# виконати команду SQL, яка додає рядок з даними
name,content в таблицю
cur.execute('INSERT INTO pages (name,content)
VALUES(?,?)', (name,content))
conn.commit() # зберегти зміни

# виконати команду SQL, яка оновлює дані
cur.execute('UPDATE pages SET content=? WHERE
name=?', (content,name))
conn.commit() # зберегти зміни
```

```
# виконати команду SQL, яка отримує результати запиту
cur.execute('SELECT content FROM pages WHERE name=?',
(name,))
#print cur.fetchone() # отримати наступний рядок
множини результату запиту, або
print cur.fetchall() # отримати усі рядки множини
результату запиту

cur.close() # закрити курсор
conn.close() # закрити базу даних
```

```
[(u'1990',)]
```

csv - читання і запис файлів CSV

CSV (Comma Separated Values) - це розповсюджений текстовий формат імпорту і експорту електронних таблиць і баз даних (наприклад з Excel). Модуль `csv` реалізує класи для читання і запису табличних даних в форматі CSV.

```
import csv
csv_file=open("some.csv", "wb") # відкрити файл для
запису
writer = csv.writer(csv_file, delimiter = ';') #
об'єкт для запису
writer.writerow([0.1,0.2,0.3]) # записати рядок
writer.writerow([0.4,0.5,0.6]) # ще один
csv_file.close() # закрити файл

csv_file=open("some.csv", "rb") # відкрити файл для
читання
reader=csv.reader(csv_file,delimiter = ';') # об'єкт
для читання
for row in reader:
```

```
    print row[0],row[1],row[2]
csv_file.close() # закрити файл
```

```
0.1 0.2 0.3
0.4 0.5 0.6
```

tarfile - читання і запис файлів архіву tar

Модуль `tarfile` дозволяє читати і записувати tar-архіви з підтримкою стиснення даних gzip або bz2. Приклад створює новий каталог, запаковує його в архів і розпаковує цей архів в інший каталог.

```
import tarfile, sys, os
encoding=sys.getfilesystemencoding() # кодування в
файловій системі (у Windows 7 - mbcs)
mydir=ur"Каталог"
mydir2=ur"Каталог2"
os.mkdir(mydir); os.mkdir(mydir2) # створити каталоги
tar = tarfile.open(ur"test_archive.tar", mode='a',
format=tarfile.PAX_FORMAT) # відкрити архів для
додання
tar.add(mydir) # додати в архів
tar.close() # закрити файл архіву

tar = tarfile.open(ur"test_archive.tar", mode='r',
format=tarfile.PAX_FORMAT) # відкрити архів для
читання
tar.extractall(path=mydir2.encode(encoding),
members=None) # розпакувати все
for x in tar.getmembers(): # для кожного елемента
архіву
    print x.name.decode(encoding) # вивести його ім'я
    print x.size # розмір в байтах
    print x.mtime # час останньої модифікації
```

```
    print x.isdir() # чи це каталог?
tar.close() # закрити файл архіву
```

```
Каталог
0
1533569368.09
True
```

zipfile - робота з ZIP-архівами

Модуль zipfile містить іструменти для створення, читання і запису ZIP-архівів. Приклад створює новий каталог, запаковує його в архів і розпаковує цей архів в інший каталог.

```
import  zipfile, os
mydir=ur"Каталог"
mydir2=ur"Каталог2"
os.mkdir(mydir); os.mkdir(mydir2) # створити каталоги
zf = zipfile.ZipFile(ur"test_archive.zip", mode='w',
compression=zipfile.ZIP_STORED) # відкрити архів для
додання без компресії (ZIP_DEFLATED - з компрессією)
for root, dirs, files in os.walk(mydir):
    zf.write(root) # додати в архів каталог
    for file in files: # для кожного файлу
        zf.write(os.path.join(root, file)) # додати в
архів файл
zf.close() # закрити файл архіву

zf = zipfile.ZipFile(ur"test_archive.zip", 'r') #
відкрити архів для читання
zf.extractall(path=mydir2, members=None, pwd=None) #
розпакувати все (підтримуються паролі pwd тільки на
розпакування і тільки ZIP2.0)
zf.close() # закрити файл архіву
```

zlib - сумісне з gzip стиснення даних

Модуль zlib містить функції для стиснення та декомпресії даних з використанням бібліотеки zlib.

```
import zlib
s="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxx"
c=zlib.compress(s, 9) # стиснути дані з найвищим
рівнем
#c=s.encode('zlib') # або так
s=zlib.decompress(c) # виконати декомпрессію
#s=c.decode('zlib') # або так
print len(s),len(c) # довжина даних до і після
стиснення
```

```
59 12
```

sys - системні параметри і функції

Модуль sys містить змінні та функції, які мають відношення до інтерпретатора Python та його середовища.

```
import sys
print sys.platform # платформа
print sys.version # версія інтерпретатора
print sys.argv # аргументи командного рядка
sys.builtin_module_names # вбудовані модулі
sys.modules # модулі що завантажуються
print sys.getsizeof(int()) # розмір об'єкта в байтах
sys.path # шлях до пошуку модулів
#sys.path.append('D:\\') # додати шлях пошуку модулів
#print sys.stdin.readline() # читати рядок з
стандартного потоку введення (тут програма буде
чекати введення)
print sys.stdin.isatty() # чи стандартний потік є
консоллю
```

```
sys.stdout.write("hello stdout\n") # запис в
стандартний потік виведення
sys.stdout=open('temp.dat','w') # перенаправлення
виведення у файл
print "hello file" # виведення у файл
sys.stdout=sys.__stdout__ # відміна перенаправлення
print "hello console" # виведення знову на консоль
try: raise IndexError # генерувати виняткову ситуацію
except: print sys.exc_info() # інформація про
виняткову ситуацію
#sys.exit() # завершення програми
```

```
win32
2.7.14 |Anaconda custom (64-bit)| (default, Oct 15 20
17, 03:34:40) [MSC v.1500 64 bit (AMD64)]
['E:\\Python_projects\\main.py']
24
True
hello stdout
hello console
(<type 'exceptions.IndexError'>, IndexError(), <trace
back object at 0x0000000005177E88>)
```

os - файлова система

Модуль os забезпечує переносимий спосіб використання функціональності, пов’язаної з операційною системою. В прикладі показані функції для роботи з файловою системою. Цю програму слід виконувати так:

```
python.exe main.py
```

```
import os,sys
os.mkdir(r'temp') # створити каталог
with open(r'temp\temp.dat','w') as f:
    f.write('hello\n') # створити файл
```

```
print os.path.isdir(r'c:\temp') # чи каталог
print os.path.isfile(r'c:\temp') # чи файл
print os.path.exists(r'temp\temp.dat') # чи існує
шлях
print os.path.getsize(r'temp\temp.dat') # розмір в
байтах
print os.path.split(r'temp\temp.dat') # розбити на
шлях і ім'я
#os.remove(r'temp\temp.dat') # видалити файл
#os.rmdir(r'temp') # видалити каталог
#print os.environ # змінні середовища (можна
змінювати)
fd=sys.stdout.fileno() # файловий дискриптор (1)
os.write(fd,'hello\n') # запис в стандартний потік як
у файл
print os.getcwd() # поточний каталог
os.chdir(r'temp') # змінити поточний каталог
cwd=os.getcwdu() # поточний каталог (unicode рядок)
print os.listdir(cwd) # список елементів каталогу
for root, dirs, files in os.walk(cwd): # або за
допомогою генератора os.walk
    print root, dirs, files
```

```
hello
True
False
True
7
('temp', 'temp.dat')
e:\python_projects
[u'temp.dat']
e:\python_projects\temp [] [u'temp.dat']
```

shutil - високорівневі операції з файлами

Модуль shutil містить високорівневі функції для операцій з файлами (копіювання, переміщення, архівування).

```
import os, shutil
os.mkdir('tmp'); os.mkdir('tmp/tmp2') # створити
каталоги
shutil.copyfile('main.py', 'tmp/tmp2/main.py') #
копіювати файл
shutil.move('tmp/tmp2', '.') # перемістити каталог в
поточний
shutil.copytree('tmp2', 'tmp3') # копіювати каталог
print shutil.make_archive('tmp/test_archive.zip',
'zip', base_dir='tmp2') # архівувати каталог
```

```
tmp/test_archive.zip.zip
```

os - створення і керування процесами

В цьому прикладі показані функції модуля os для створення і керування процесами. Ознайомтесь також з більш новим модулем subprocess.

```
import os
print os.getpid() # ідентифікатор процесу
os.system(r'start calc.exe') # виконує команду
оболонки
print os.system(r'echo hello') # виконує команду
оболонки
print os.popen(r'echo world').read() # читати
результати команди оболонки
id = os.spawnv(os.P_NOWAIT,
'c:\\Windows\\Notepad.exe',[r'
c:\Python27\README.txt']) # виконує програму без
очікування виходу з неї
status = os.waitpid(id, 0) # але тут чекає завершення
```

```
процесу id
print 'status=', status
os.startfile(r'c:\Python27\README.txt') # виконує
файл відповідним застосуванням
os.execl(r'c:\Windows\Notepad.exe', '
c:\Python27\README.txt') # виконує файл, замінює
поточний процес
print "hello" # ця команда вже не виконається
```

```
5144
hello
0
world

status= (820, 0)
```

subprocess - керування підпроцесами

Процес - це об’єкт операційної системи, який описує програму, що виконується. Процес є контейнером, який містить такі ресурси як ідентифікатор процесу, образ виконуваного машинного коду програми, пам’ять, дескриптори ресурсів ОС, атрибути безпеки, стан процесора, потоки процесу. Модуль subprocess дозволяє створювати нові процеси, під’єднуватись до їх input/output/error каналів та отримувати їхні коди завершення. Цей модуль призначений для заміни кількох старих модулів і функцій (`os.system`, `os.spawn*`, `os.popen*`, `popen2.*`, `commands.*`).

```
import subprocess
p=subprocess.Popen(['notepad',
r'c:\Python27\README.txt']) # повертає об'єкт Popen,
який являє собою новий процес
print p.wait() # чекає його завершення, повертає код
завершення
```

```
#print subprocess.call(r'notepad
c:\Python27\README.txt') # те саме
print subprocess.call('ver', shell=True) # те саме в
консолі
print subprocess.check_output('python -c "x=1\nprint
x"') # виконує команду і повертає її виведення
# або
p = subprocess.Popen('python -c "print 1+1"',
stdout=subprocess.PIPE)
print p.stdout.read()

p = subprocess.Popen('python', stdin=subprocess.PIPE,
stdout=subprocess.PIPE, stderr=subprocess.PIPE) #
новий процес
out, err = p.communicate("print 1+2") # посилає дані
в stdin процесу
print out, err # і читає дані з stdout і stderr
```

```
0
Microsoft Windows [Version 6.1.7601]
0
1
2
3
```

subprocess - міжпроцесова взаємодія

main.py - модуль клієнта

Міжпроцесова взаємодія (англ. IPC) - це обмін даними між процесами. Як правило реалізується засобами ОС. До методів IPC належать: файли, неіменовані і іменовані канали, черги повідомлень, сигнали, спільна пам’ять, сокети і файли, що відображаються в пам’ять. В прикладі створюється канал між стандартними потоками введення/виведення/помилок

(stdin/stdout/stderr) процесів. Цей модуль створює новий процес server.py, відсилає йому дані на stdin та отримує дані з його stdout.

```
import subprocess, pickle
data=['A','B','C'] # дані
s = pickle.dumps(data) # серіалізувати список в рядок
s=s.encode("string_escape") # перетворити в рядковий
літерал Python (без "\n")
p=subprocess.Popen(["python", "server.py"],stdin =
subprocess.PIPE, stdout= subprocess.PIPE, stderr=
subprocess.PIPE) # створити процес
stdout, stderr = p.communicate(input=s) # надіслати
дані в stdin, отримати дані з stdout, чекати
завершення процесу
s=stdout.decode("string_escape") # перетворити з
рядкового літералу Python
print pickle.loads(s) # перетворити в список
```

```
['A', 'B', 'C', 'D']
```

server.py - модуль сервера

Модуль отримує дані від клієнта через stdin та відсилає їх назад через stdout.

```
import pickle,sys
# ! тут заборонено використовувати print
s=sys.stdin.read().decode("string_escape") #
перетворити з рядкового літералу Python
data = pickle.loads(s) # перетворити в список
data.append('D') # додати дані
s=pickle.dumps(data) # серіалізувати список в рядок
s=s.encode("string_escape") # перетворити в рядковий
літерал Python (без "\n")
sys.stdout.write(s) # записати в stdout
```

thread - створення багатьох потоків керування

Потоком виконання називають частину процесу, яка може виконуватись паралельно з іншими потоками цього процесу і використовувати спільні з ними ресурси. Синхронізація потоків і процесів - це механізм, який перешкоджає одночасному їх зверненню до спільно використовуваних ресурсів. Модуль thread забезпечує низькорівневі (на відміну від threading) примітиви для роботи з багатьма потоками. В прикладі створюються 4 потоки, які виконують функцію f. Звернення потоків до спільного списку A синхронізовано за допомогою простого об'єкта блокування allocate_lock. Нижче показані результати роботи програми з цим об'єктом і без нього. Зауважте, що в CPython існує глобальне блокування інтерпретатора Global Interpreter Lock (GIL), яке являє собою механізм синхронізації потоків, що не дозволяє в один момент часу виконуватись більше ніж одному потоку. Тому застосовуйте модуль multiprocessing, якщо програмі потрібно задіяти для обчислень кілька процесорів. А багатопотоковість краще застосовувати у випадку багатьох одночасних задач введення-виведення.

```
import thread,time

def f(i): # функція виконується в окремому потоці
    mutex.acquire() # блокувати (лише один потік може
виконуватись в один і той самий момент часу)
    A.append(i)
    time.sleep(1)
    A.append(i)
    mutex.release() # розблокувати
    T[i]=1 # повідомити головному потоку, що потік
завершився
A=[] # глобальний список
T=[0,0,0,0] # глобальний список (якщо потік `i`
```

```
завершився, то T[i]=1)
mutex = thread.allocate_lock() # створити блокуючий об'єкт
for i in range(4): # створити 4 потоки
    thread.start_new(f, (i, )) # стартувати потік 'i'
while 0 in T: # поки усі потоки не приєднаються
    pass # тут головний потік може робити щось своє
print A
```

```
[0, 0, 1, 1, 2, 2, 3, 3]
[0, 1, 2, 3, 1, 3, 2, 0]
```

threading - високорівневий інтерфейс потоків

Цей модуль створює високорівневі інтерфейси потоків на основі низькорівневого модуля `thread`. Потоки описуються нащадком класу `threading.Thread`, а їх активність - перевизначеним методом `run`. В прикладі створюються 4 потоки, які виконують код в методі `run`. Звернення потоків до спільного списку `A` синхронізовано за допомогою простого об'єкта блокування `threading.Lock`. Нижче показані результати роботи програми з цим об'єктом і без нього. Додатково створюється потік, який стартує через 2 секунди і додає в список `A` рядок 'timer'.

```
import threading, time

class Thread(threading.Thread): # успадкований від threading.Thread
    def __init__(self, i): # конструктор
        self.i=i # ідентифікатор потоку
        threading.Thread.__init__(self) # виклик конструктора базового класу
    def run(self): # забезпечує логіку потоку
        mutex.acquire() # блокувати (лише один потік може виконуватись в один і той самий момент часу)
        #semaphore.acquire() # або так
```

```
        A.append(self.i)
        time.sleep(1)
        A.append(self.i)
        mutex.release() # розблокувати
        #semaphore.release() # або так

mutex = threading.Lock() # те саме що
thread.allocate_lock()
#semaphore=threading.Semaphore(1) # або семафор
(тільки 1 потік одночасно)
A=[] # глобальний список
T=[] # список потоків
for i in range(4): # створити 4 потоки
    t=Thread(i) # створити потік
    t.start() # виконати метод run в потоці
    T.append(t) # додати в список потоків
t = threading.Timer(2.0, lambda: A.append('timer')) #
створити потік,
t.start() # який стартує через 2 с
T.append(t)
for t in T:
    t.join() # поки усі потоки не приєднаються
print A
```

```
[0, 0, 1, 'timer', 1, 2, 2, 3, 3]
[0, 1, 2, 3, 2, 3, 1, 0, 'timer']
```

multiprocessing - підтримка багатох процесів

multiprocessing - це пакет, який підтримує створення процесів з використанням API, який подібний на API модуля threading. Забезпечує локальний і віддалений паралелізм, ефективно долає GIL шляхом використання підпроцесів замість потоків. В приклад розпаралелюється задача знаходження квадратів 50 матриць 1000x1000 за допомогою класу `Poll` і його методу `map`. Зауважте, що виграш в продуктивності буде досягнуто тільки на

багатопроцесорній системі. Виконайте програму послідовно в паралельному і звичайному режимах так:

```
python main.py
python main.py s
```

Для визначення продуктивності програми використано модуль `timeit`. В Windows можна також використати команду:

```
echo %time% & main.py & call echo %^time%
```

```
import numpy as np
from multiprocessing import Pool # задіяти багато
процесів
#from multiprocessing.dummy import Pool # або задіяти
багато потоків
import sys, timeit
def f(x): # функція, яка виконується в кожному
процесі
    return x*x
if __name__ == '__main__':
    time = timeit.default_timer()
    X=[np.matrix(np.random.rand(1000,1000)) for i in
range(50)] # 50 матриць 1000x1000
    if 's' in sys.argv:
        Y=map(f,X) # застосувати f для кожного у X
(тільки 1 процес)
    else:
        p=Pool(4) # створити пул 4-х процесів
        Y=p.map(f, X) # задіяти 4 процеси
    print timeit.default_timer() - time
```

```
9.77497753973
15.1325679658
```

multiprocessing - запуск паралельних задач

Аналог прикладу concurrent.futures на основі multiprocessing. Тут функціям `ProcessPoolExecutor`, `submit`, `result`, `map` відповідають `Pool`, `apply_async`, `get`, `map`.

```
import time
from multiprocessing import Pool
def f(x): # функція, яка буде виконуватись в окремих процесах
    time.sleep(x) # затримка (тільки для тестування паралельності)
    return x
if __name__ == '__main__':
    pool = Pool() # пул процесів
    a = pool.apply_async(f, [4]) # AsyncResult
    b = pool.apply_async(f, [2])
    while any([a,b]): # отримати результати асинхронно
        if a and a.ready(): print a.get(); a=False
        if b and b.ready(): print b.get(); b=False
    #print pool.map(f, [1,2]) # або чекати усі результати
```

```
2, 4
```

multiprocessing - міжпроцесова взаємодія

Для обміну об'єктами між процесами можна використовувати черги (`Queue`), канали (`Pipe`), спільну пам'ять (`Value`, `Array`). Клас `Manager` створює об'єкт, що контролює серверний процес, який зберігає об'єкти Python і дозволяє іншим процесам використовувати їх. Використання об'єктів `Manager` більш гнучке, ніж об'єктів спільної пам'яті, так як вони можуть підтримувати об'єкти довільних типів. Але вони більш повільні. У прикладі за допомогою

Manager створюється список, у який базовий і дочірній процес паралельно додають елементи.

```
from multiprocessing import Process, Manager
def f(L): # функція, що виконується в окремому процесі
    L.append(4)
if __name__ == '__main__':
    manager = Manager() # менеджер
    L = manager.list([1,2]) # спільний для процесів список
    p = Process(target=f, args=(L, )) # новий процес
    p.start() # стартувати процес (робота з L розпаралелюється)
    L.append(3)
    p.join() # приєднати процес
    print L
```

```
[1, 2, 3, 4]
```

socket - низькорівневий мережевий інтерфейс

server.py - модуль сервера

Модуль `socket` забезпечує доступ до API сокетів BSD. Доступний для багатьох сучасних ОС. Найчастіше використовуються такі функції як `socket` (створити кінцеву точку з'єднання), `bind` (присвоїти сокету адресу), `listen` (режим очікування вхідних повідомлень), `accept` (прийняти з'єднання), `connect` (установлює з'єднання), `send` (надсилає дані), `recv` (приймає дані). Нижче наведено модуль однопотокового сервера з адресою '127.0.0.1' і портом 50007. Сервер отримує дані через мережу від клієнтів та відсилає їх назад. Для виконання прикладу не потрібно наявності віддаленої машини, так як сервер і клієнт будуть виконуватись на одній машині (адреса '127.0.0.1' або 'localhost' або '' означає цей комп'ютер). Виконайте цей модуль командою

`python server.py`, а в іншому консольному вікні введіть `python client.py`.

```
import socket
from socketFileIO import write, read
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# відкрити сокет типу TCP/IP
s.bind(('localhost', 50007)) # зв'язати сокет з
локальною адресою і портом
s.listen(1) # дозволити не більше 1 з'єднання з
клієнтом
while True: # цикл
    soc, addr = s.accept() # чекає з'єднання з
клієнтом, повертає об'єкт сокета та адресу і порт
клієнта
    print 'Server is connected to client', addr
    x=soc.recv(255) # отримати з сокета рядок
довжиною не більше 255
    #x=read(soc) # або
    print 'Client:', x
    soc.sendall(x) # надіслати рядок
    #write(soc,x) # або
    soc.close() # закрити з'єднання з клієнтом
    if x=='End': break # якщо отримано такий рядок,
то завершити цикл
```

```
Server is connected to client ('127.0.0.1', 50827)
Client: A
Server is connected to client ('127.0.0.1', 50828)
Client: B
Server is connected to client ('127.0.0.1', 50829)
Client: C
Server is connected to client ('127.0.0.1', 50830)
Client: End
```

client.py - модуль клієнта

Надсилає дані через мережу серверу з адресою '127.0.0.1' і портом 50007 та отримує їх назад.

```
import socket
from socketFileIO import write, read
for x in ['A','B','C','End']: # дані, що будуть відсилатись
    s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM) # відкрити сокет типу TCP/IP
    s.connect(('127.0.0.1', 50007)) # з'єднати сокет з сервером
    s.sendall(x) # надіслати рядок
    #write(s,x) # або
    x=s.recv(255) # отримати рядок
    #x=read(s) # або
    print 'Server:', x
    s.close() # закрити сокет
```

```
Server: A
Server: B
Server: C
Server: End
```

socketFileIO.py - читання і запис об'єктів Python через сокет

Нижче наведено код модуля `socketFileIO.py` з функціями `write` і `read`, які дозволяють відсилати чи отримувати об'єкти Python по мережі через сокети. В модулях `server.py` і `client.py` розкоментуйте виклики цих функцій і закоментуйте виклики `sendall` та `recv`.

```
import pickle

def write(soc, obj):
```

```
    "Відсилає об'єкт obj через сокет soc"
    f = soc.makefile('wb') # створити файл, 
асоційований з сокетом
    pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL) # 
серіалізувати obj в файлі
    f.close() # закрити файл

def read(soc):
    "Повертає об'єкт, отриманий з сокета soc"
    f = soc.makefile('rb') # створити файл, 
асоційований з сокетом
    obj = pickle.load(f) # десеріалізувати obj з 
файлу
    f.close() # закрити файл
    return obj
```

SocketServer - каркас для мережевих серверів

Високорівневий модуль `SocketServer` спрощує задачі створення мережевих серверів. Для створення власного обробника мережевих запитів потрібно успадкувати клас `BaseRequestHandler` і перевизначити метод `handle`. В прикладі на основі `SocketServer` створено багатопотоковий сервер з адресою ‘127.0.0.1’ і портом 50007. Сервер отримує дані через мережу від клієнтів та відсилає їх назад. Для тестування багатопотоковості запустіть цей сервер `python serverT.py` і кілька клієнтів `python client.py`. В диспетчері завдань Windows 7 можна побачити, як змінюється кількість потоків процесу сервера.

```
import SocketServer, time
from socketFileIO import write, read  # якщо потрібно

class 
MyClientHandler(SocketServer.BaseRequestHandler): # 
клас обробника запитів
    def handle(self): # обробляє запити
```

```
(перевизначений)
        print self.client_address # показати адресу
клієнта
        x=self.request.recv(255) # отримати рядок
        #x=read(self.request) # або
        print 'Client:', x
        time.sleep(10) # затримка (для тестування
багатопотоковості)
        self.request.sendall(x) # надіслати рядок
        #write(self.request,x) # або
        self.request.close() # закрити з'єднання з
клієнтом

# створити багатопотоковий TCP сервер з обробником
MyClientHandler
server = SocketServer.ThreadingTCPServer(('', 50007),
MyClientHandler) # порт 50007 або порт 0 (довільний
незайнятий порт)
try:
    server.serve_forever() # обробляти запити вічно
except KeyboardInterrupt: # якщо натиснуто Ctrl-C
    server.shutdown() # зупинити сервер
```

CGI HTTP сервер

Веб-сервер — це програма, яка приймає HTTP-запити від клієнтів (зазвичай веб-браузерів) і видає їм HTTP-відповіді, як правило, з HTML-сторінкою. Протокол передачі гіпертексту HTTP описує HTTP-повідомлення, які складаються з стартового рядка (тип повідомлення), заголовків (параметри транзакції HTTP) і необов'язкового тіла (наприклад з даними HTML). HTTP-повідомлення можна переглянути, наприклад, в браузері Firefox 61 в меню веб-розробка/мережа. Приклад HTTP-запиту типу GET до ресурсу /hello.html.

```
GET /hello.html HTTP/1.1
Host: localhost
(пустий рядок)
```

Метод GET використовується для запиту вмісту вказаного ресурсу, а метод POST - для передачі даних вказаному ресурсу. Приклад HTTP-відповіді сервера з кодом стану 200 (виконано):

```
HTTP/1.0 200 OK
Content-type: text/html
(пустий рядок)
<html><body>Hello</body></html>
```

В прикладі створено `BaseHTTPServer.HTTPServer` сервер з підтримкою запитів GET, HEAD, POST і CGI-програм. В даному випадку усі CGI-програми повинні бути розташовані в каталозі cgi поряд з сервером. Запустіть сервер та в адресному рядку браузера введіть:

```
http://localhost/hello.html
```

```
import os, sys
import BaseHTTPServer, CGIHTTPServer
with open('hello.html', 'w') as f: # створити
документ HTML
    f.write("<html><body>Hello</body></html>")
class Handler(CGIHTTPServer.CGIHTTPRequestHandler): #
обробник запитів
    cgi_directories = ["/cgi"] # каталог з CGI-
програмами
srvraddr = ('localhost', 80) # ім'я хоста, номер
порта
srvrobj = BaseHTTPServer.HTTPServer(srvraddr,
Handler) # сервер
srvrobj.serve_forever() # обслуговувати клієнтів до
завершення
```



Рисунок 2 - Результати роботи сервера

CGI-програма simple.py - генерація форми запиту

CGI (Common Gateway Interface) - стандартний протокол для взаємодії програми веб-сервера із зовнішньою консольною програмою (CGI-програмою або шлюзом). Після запиту клієнта CGI-програма виконується сервером в окремому процесі, обробляє дані запиту і генерує відповідь сервера. Будь-яка CGI-програма повертає в стандартний потік виведення заголовок HTTP, пустий рядок і дані. Запустіть сервер та в адресному рядку браузера введіть:

```
http://localhost/cgi/simple.py
```

Для тестування методу GET справте нижче `method="post"` на `method="get"`.

```
html="""<html><body>
<form action="/cgi/get_post.py" method="post">
First Name: <input type="text" name="first_name"><br
/>
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" /></form>
</body></html>"""
print"Content-type: text/html\r"# заголовок
print"\r"# пустий рядок
print html # дані
```

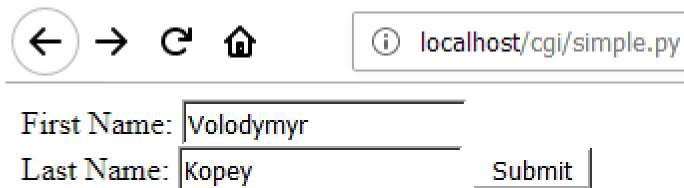


Рисунок 3 - Результати роботи CGI-програми simple.py

CGI-програма get_post.py - обробка запитів GET і POST

CGI-програма може отримати доступ до рядка запиту (даних форми) за допомогою `cgi.FieldStorage`. Запустіть сервер та в адресному рядку браузера введіть для тестування методів GET і POST, відповідно:

```
http://localhost/cgi/get_post.py?first_name=Volodymyr
&last_name=Kopey
http://localhost/cgi/get_post.py`
```

Або, якщо форма розташована у файлах HTML, відповідно:

```
http://localhost/GET.html
http://localhost/POST.html
```

```
import cgi # модуль для обробки cgi
form = cgi.FieldStorage() # об'єкт FieldStorage
first_name = form.getvalue('first_name') # дані з
першого поля
last_name  = form.getvalue('last_name') # дані з
другого поля
print"Content-type: text/html\r\n\r\n"# заголовок,
пустий рядок
print"<h2>Hello %s%s</h2>"% (first_name, last_name) #
дані
```

localhost/cgi/get_post.py

Hello Volodymyr Kopey

Рисунок 4 - Результати роботи CGI-програми get_post.py

WSGI сервер

WSGI (Web Server Gateway Interface) - стандарт взаємодії між Python-програмою і самим веб-сервером. За стандартом WSGI веб-програма повинна бути об’єктом, що викликається, і приймати два параметра: словник змінних середовища (environ) і обробник запиту (start_response). Модуль `wsgiref.simple_server` реалізує простий HTTP-сервер, який виконує одну WSGI-програму. Запустіть сервер та в адресному рядку браузера введіть:

```
http://localhost/?name=Volodymyr
http://localhost
```

```
from wsgiref.simple_server import make_server
from cgi import parse_qs, escape
from PIL import Image
import StringIO

html="""<html><body><form method="post">
Name: <input type="text" name="name">
<input type="submit" value="Submit"
/></form></body></html>"""

def application(environ, start_response):
    # вивести вміст деяких змінних середовища
    print 'QUERY_STRING:', environ['QUERY_STRING']
    print 'REQUEST_METHOD:',
environ['REQUEST_METHOD']
    print 'PATH_INFO:', environ['PATH_INFO']
    print 'HTTP_ACCEPT:', environ['HTTP_ACCEPT']
```

```
    response_headers=[('Content-Type', 'text/html')]
# заголовки
    if environ['REQUEST_METHOD'] == 'GET': # якщо
запит GET
        parameters=parse_qs(environ['QUERY_STRING'])
# парам. з рядка запиту
        if 'name' in parameters: # якщо в запиті є
параметр 'name'
            name = escape(parameters['name'][0]) #
його значення
            response_body="<h2>Hello %s </h2>" %
(name) # тіло відп.
        elif environ['PATH_INFO']=="/pic.png": # якщо
запит на рисунок
            image = Image.new('RGB', (10, 10), (0,
255, 0)) # рисунок
            out=StringIO.StringIO()
            image.save(out, format='PNG') # зберегти
в пам'ять
            response_headers = [('Content-Type',
'image/png')] # заголовки
            response_body=out.getvalue() # тіло відп.
(дані рисунка)
        else: # якщо інший запит
            response_body=html # тіло відп. (документ
HTML з формою)

    if environ['REQUEST_METHOD'] == 'POST': # якщо
запит POST
        try: # змінна CONTENT_LENGTH може бути пуста
або відсутня
            request_body_size =
int(environ.get('CONTENT_LENGTH', 0))
        except (ValueError):
```

```
            request_body_size = 0
        request_body =
environ['wsgi.input'].read(request_body_size) # тіло
запиту, передане через форму
        parameters = parse_qs(request_body) # словник
параметрів форми
        name = escape(parameters['name'][0]) #
значення параметра 'name'
        response_body='<h2><img src="pic.png"
alt="pic">Hello %s </h2>' % (name) # тіло відповіді

    start_response('200 OK', response_headers)
    return [response_body]

httpd = make_server('localhost', 80, application) #
WSGI сервер
httpd.serve_forever()
```

```
QUERY_STRING: name=Volodymyr
REQUEST_METHOD: GET
PATH_INFO: /
HTTP_ACCEPT: text/html,application/xhtml+xml,applicat
ion/xml;q=0.9,*/*;q=0.8
```



Hello Volodymyr

Рисунок 5 - Відповідь на запит GET

```
QUERY_STRING:
REQUEST_METHOD: GET
PATH_INFO: /
HTTP_ACCEPT: text/html,application/xhtml+xml,applicat
ion/xml;q=0.9,*/*;q=0.8
```

Рисунок 6 - Форма для запиту POST

```
QUERY_STRING:
REQUEST_METHOD: POST
PATH_INFO: /
HTTP_ACCEPT: text/html,application/xhtml+xml,applicat
ion/xml;q=0.9,*/*;q=0.8

QUERY_STRING:
REQUEST_METHOD: GET
PATH_INFO: /pic.png
HTTP_ACCEPT: */*
```

Рисунок 7 - Відсилання рисунку

urllib2 - запити до HTTP серверів

urllib2 модуль містить функції і класи, які допомагають отримувати інформацію за URL переважно від HTTP серверів. Підтримується аутентифікація, переадресація, cookie, проксі-сервера та інше.

```
import urllib, urllib2
url="http://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D1%81%
D1%96%D0%B2"
url=urllib2.unquote(url).decode("utf-8") #
перетворити URL в легкий для читання формат
print url
```

```
print urllib2.quote(url.encode("utf-
8"),safe="%/:=&?~#+!$,;'@()*[]") # перетворити назад

response =
urllib2.urlopen('http://httpbin.org/get?name=John') #
отримати відповідь за HTTP GET запитом з параметром
name=John
#print response.info() # заголовки відповіді у
вигляді словника
print response.info()['Content-Type'] # тип тіла
відповіді
print response.read(1) # читати 1 байт тіла відповіді
response.close() # закрити файл

form_data = urllib.urlencode({'name':'John'}) # дані
для відправлення
headers={'User-Agent' : 'Mozilla 5.0'} # заголовки
запиту
request = urllib2.Request('http://httpbin.org/post',
form_data, headers) #  HTTP POST запит з даними
form_data і заголовками headers
response = urllib2.urlopen(request) # отримати
відповідь
print response.read(1) # читати 1 байт тіла відповіді
```

```
http://uk.wikipedia.org/wiki/Косів
http://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D1%81%D1%96
%D0%B2
application/json
{
{
```

xml.dom.minidom - мінімальна реалізація DOM

XML - це стандарт побудови мов розмітки (мов, що використовують спеціальні анотації для розмітки тексту) ієрархічно

структурованих даних. DOM (Document Object Model) - це незалежний від мови програмування програмний інтерфейс, який дозволяє створювати, читати і змінювати XML документи. DOM подає XML документи як деревовидну структуру, де кожен вузол є об'єктом, що відповідає частині документу. xml.dom.minidom - це мінімальна реалізація інтерфейсу DOM, який подібний на ті, що використовуються в інших мовах. Вона простіша ніж повний DOM і суттєво менша.

```
from xml.dom import minidom

################# Document Objects
(minidom.Document()) ##########

# створити XML документ з кореневим тегом 'html'
doc=minidom.getDOMImplementation().createDocument(Non
e, 'html', None)
html=doc.documentElement # кореневий елемент

# або створити XML документ так:
#doc=minidom.Document() # XML документ
#html=doc.createElement("html")# створити кореневий
елемент (тільки один)
#doc.appendChild(html) # додати дочірній вузол (тут
doc - вузол)

body = doc.createElement('body') # створити елемент
html.appendChild(body) # додати дочірній вузол до
html

div = doc.createElement('div') # створити елемент
txtNode=doc.createTextNode('Text') # створити
текстовий вузол
#print txtNode.data # вміст текстового вузла
div.appendChild(txtNode) # додати дочірній вузол до
```

```
div
body.appendChild(div) # додати дочірній вузол до body

################## Element Objects
(minidom.Element()) ###########

elements=doc.getElementsByTagName('div') # знайти усі
елементи з тегом div
#print elements[0].toxml() # вивести перший з них в
форматі XML
el=div # елемент div
#print el.tagName # ім'я тегу
el.setAttribute('id', '1') # задати атрибути
el.setIdAttribute('id') # задати ID атрибут (для
getElementById())
#print el.hasAttribute('id') # чи має атрибут 'id'
#print el.getAttribute('id') # значення атрибута 'id'
#print el.getAttributeNode('id') # вузол атрибута
el.removeAttribute('id')
el.setAttribute('id', '1')
el=doc.getElementById('1') # знайти елемент з ID='1'

################# Node Objects (minidom.Node())
##################

node=div # вузол елемента div
#print node.nodeName # ім'я вузла (div)
#print node.nodeType # тип вузла (1 - ELEMENT_NODE)
#print node.nodeValue # текстове значення вузла
#print node.hasChildNodes() # чи має підвузли
#print node.parentNode # батьківський вузол
#print node.nextSibling # наступний споріднений
#print node.previousSibling # попередній споріднений
#print node.childNodes # дочірні вузли
#print node.firstChild # перший дочірній
```

```
#print node.lastChild # останній дочірній
#print node.hasAttributes() # чи має атрибути
#print node.attributes['id'].nodeValue # значення
атрибута id
#print node.isSameNode(div) # чи це той самий вузол
clon=node.cloneNode(True) # клонувати з підвузлами
body.insertBefore(clon, div) # вставити дочірній
перед div
body.removeChild(clon) # видалити дочірній clon
body.appendChild(clon) # додати дочірній
body.removeChild(clon)

######################## xml.dom.minidom
###########################
print doc.toprettyxml(' ') # вивести в форматі з
відступами
f=open("my.html","w") # відкрити файл для запису
f.write(doc.toprettyxml(' ')) # зберегти документ
f.close()

doc2 = minidom.parse('my.html') # читати XML документ
з файлу
doc3=minidom.parseString('<A>x</A>') # читати XML
документ з рядка
#print doc3.toxml() # вивести документ в форматі XML
```

```
<?xml version="1.0" ?>
<html>
  <body>
   <div id="1">Text</div>
  </body>
</html>
```

xml.etree.ElementTree - ElementTree XML API

Модуль містить визначення типу Element - гнучкого контейнера, який призначений для зберігання ієрархічних структур даних в пам’яті. Використовується для роботи з XML і іншими деревовидними даними. Кожен елемент має такі властивості як тег, набір атрибутів, тестовий рядок, хвостовий рядок, дочірні елементи.

```
import xml.etree.ElementTree as ET
#ET.VERSION # версія бібліотеки
root = ET.Element("root") # створити кореневий
елемент
#root = ET.XML("<root></root>") # або створити з
тексту
print ET.tostring(root) # текст елемента
("<root></root>")
print root.tag # тег елемента
root.append(ET.Element("one")) # додати піделемент з
таким тегом
two=ET.SubElement(root, "two") # або так
two.attrib["first"] = "1" # створити атрибут елемента
two.text = "text" # текст в елементі
two_one=ET.SubElement(two, "two_one") # додати
піделемент
two_one.tail="text" # текст після елемента
root.insert(0, ET.Element("zero")) # вставити елемент
в позицію
root.remove(root.find("zero")) # знайти перший
піделемент з таким тегом і видалити його
es=root.findall("one") # знайти всі піделементи з
таким тегом
es=root.findall(".//one") # знайти за шаблоном:
#'tag' - відповідає елементам верхнього рівня з тегом
tag
#'parent/tag' - відповідає елементам з тегом tag,
якщо вони дочірні для parent
```

```
#'*' - будь-які дочірні елементи
#'.' - починає пошук з поточного вузла
#'//' - відповідає всім вкладеним елементам на всіх
рівнях нижче рівня вказаного елемента
txt=root.findtext("two") # знайти текст першого
піделемента з таким тегом
print len(root) # кількість піделементів
print root[1].tag # тег другого елемента
print root[1].attrib  # атрибути другого елемента
(словник)
nodes = root[:] # усі піделементи або
root.getchildren()
for node in root: # цикл по піделементам
    print node.tag

print ET.tostring(root) # вивести як XML

tree = ET.ElementTree(root) # дерево елементів
tree.write("page.xml") # зберегти у файл

tree2 = ET.ElementTree() # або відразу
ET.ElementTree("page.xml")
tree2.parse("page.xml") # читати з файлу
subel=tree2.getroot()[0] # перший піделемент
subtree=ET.ElementTree(subel) # піддерево

for parent in tree2.getiterator(): # показати все
дерево
    #або getiterator("tagname") - для заданих тегів
    print parent.tag,
#    for child in parent:
#        print ' '*5+child.tag
```

```
<root />
root
```

```
2
two
{'first': '1'}
one
two
<root><one /><two first="1">text<two_one />text</two>
</root>
root one two two_one
```

HTMLParser - простий парсер HTML і XHTML

Цей модуль визначає клас HTMLParser, який служить як основа для синтаксичного аналізу файлів HTML і XHTML. Для парсингу необхідно створити похідний від HTMLParser клас і перевизначити його методи. У Python 2.7 працює також з некоректними html. Для високопродуктивного парсингу використовуйте lxml (з ElementTree API) або Beautiful Soup.

```
from HTMLParser import HTMLParser

class MyHTMLParser(HTMLParser): # успадковує HTMLParser і перевизначає його методи, шукає дані усіх тегів <p>
    def __init__(self): # конструктор
        HTMLParser.__init__(self)
        self.intag = False # в середині тегу?
        self.data = [] # список знайдених даних
    def handle_starttag(self, tag, attrs):
        "Викликається коли знайдено початковий тег (наприклад <p>)"
        print "Початковий тег", tag
        print "Атрибути", attrs
        if tag=='p': # якщо тег p
            self.intag=True # знаходимось всередині тегу
    def handle_endtag(self, tag):
```

```
        "Викликається коли знайдено кінцевий тег
(наприклад </p>)"
        print "Кінцевий тег:", tag
        if tag=='p': # якщо тег p
            self.intag=False # знаходимось поза тегом
p
    def handle_data(self, data):
        "Викликається коли знайдено дані ...
(наприклад <p>...</p>)"
        print "Дані:", data
        if self.intag: # якщо в середині тегу p
            self.data.append(data) # додати в список
результатів дані

parser = MyHTMLParser() # об'єкт класу
html="""<html><body>
<p align="justify">Текст</p>
<a href="index.html">Індекс</a>
</body></html>""" # документ HTML для парсингу
parser.feed(html) # виконати парсинг
parser.close()
print "\nЗнайдені дані:", parser.data[0]
#parser.handle_starttag('a', [('href',"index.html")])
```

```
Початковий тег html
Атрибути []
Початковий тег body
Атрибути []
Дані:

Початковий тег p
Атрибути [('align', 'justify')]
Дані: Текст
Кінцевий тег: p
Дані:
```

```
Початковий тег a
Атрибути [('href', 'index.html')]
Дані: Індекс
Кінцевий тег: a
Дані:

Кінцевий тег: body
Кінцевий тег: html

Знайдені дані: Текст
```

Tkinter - проста програма з графічним інтерфейсом

Модуль `Tkinter` - це інтерфейс до Tcl/Tk (скриптової мови Tcl та її бібліотеки Tk) для мови Python. Використовується для створення кросплатформних програм з графічним інтерфейсом (GUI). Якщо не потрібно, щоб програма показувала DOS вікно, змініть її розширення з .py на .pyw.

```
from Tkinter import * # імпортувати все з модуля
Tkinter
def Button1Click(): # функція, яка викликається під
час натиску на Button1
    x=float(s.get()) # присвоїти `x` значення `s`
    s.set(x**2) # установити `s` значення x**2
root = Tk() # головне вікно програми
root.title('Simple GUI app') # надпис на вікні
root.resizable(width=TRUE, height=FALSE) # дозволити
зміну розміру вікна по ширині
root.geometry("200x150+0+0") # розмір вікна
Button1=Button(root, text="SQR",
command=Button1Click) # створити кнопку, пов'язати з
функцією command1
Button1.place(relx=0.6, rely=0.5, relwidth=0.3,
relheight=0.1) # розташувати на вікні
```

```
#Button1.pack(side=RIGHT) # або розташувати на вікні
справа
s=StringVar() # створити рядкову змінну
Entry1 = Entry(root,textvariable=s,width=10) #
створити поле вводу, пов'язати зі змінною s
Entry1.place(relx=0.1, rely=0.5, relwidth=0.3,
relheight=0.1) # розташувати на вікні
s.set(0) # установити рядковій змінній значення 0
root.mainloop() # головний цикл програми (для обробки
подій)
```

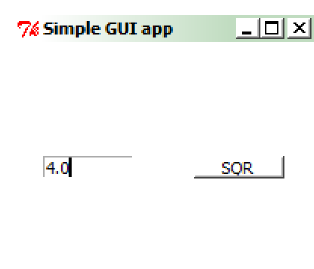


Рисунок 8 - Вікно програми

Tkinter - основні класи

В прикладі показано використання основних класів Tkinter для створення програм з графічним інтерфейсом. Використано такі класи як Tk (головне вікно), Frame (фрейм або прямокутна область на екрані), Button (кнопка), Label (надпис), Entry (текстове поле), Checkbutton (прапорець), Radiobutton (перемикач), Listbox (список), Canvas (канва або область для рисування), Scale (шкала), Menu (меню), StringVar (текстова змінна), IntVar (ціла змінна), BooleanVar (булева змінна), DoubleVar (дійсна змінна).

```
from Tkinter import *
class MyFrame(Frame): # клас, успадкований від Frame
```

```
    def __init__(self, master=None): # конструктор
        Frame.__init__(self, master) # виклик
конструктора базового класу
        self.grid() # розмістити фрейм
        self.button1 = Button(self, text="Button",
command=self.command1) # створити кнопку, встановити
її властивості
        # або встановити властивості так:
        self.button1["text"] = "Button" # надпис
        self.button1["command"] =  self.command1 #
метод для виконання
        # або встановити властивості так:

self.button1.config(text="Button",command=self.comman
d1)
        self.button1.grid(row=0, column=0) #
розмістити в рядку 0 і стовпчику 0
        self.label1=Label(self,text="Label") #
створити надпис
        self.label1.grid(row=0, column=1) #
розмістити
        self.tv=StringVar() # створити рядкову змінну
        self.entry1=Entry(self,textvariable=self.tv)
# створити текстове поле, пов'язати з змінною tv
        self.entry1.insert(0, 3.14) # вставити текст
(або так: self.tv.set("3.14"))
        self.entry1.grid(row=0, column=2) #
розмістити
        self.bv=BooleanVar() # створити булеву змінну

self.check1=Checkbutton(self,variable=self.bv) #
створити прапорець, пов'язати зі змінною bv
        self.check1.grid(row=0, column=3) #
розмістити
        self.iv=IntVar() # створити цілу змінну
```

```
        self.radio1=Radiobutton(self,
text='Radio1',variable=self.iv, value=1) # створити
перемикач, пов'язати з змінною iv
        self.radio1.grid(row=1, column=0) #
розмістити
        self.radio2=Radiobutton(self,
text='Radio2',variable=self.iv, value=2) # створити
перемикач, пов'язати зі змінною iv
        self.radio2.grid(row=1, column=1) #
розмістити
        self.iv.set(2) # установити значення 2
(включити другий перемикач)
        self.list1=Listbox(self) # створити список
        self.list1.grid(row=1,
column=2,rowspan=5,columnspan=1) # розмістити
        for x in ["Red","Blue","Green"]: # заповнити
список
            self.list1.insert(END,x)
        self.list1.selection_set(0) # вибрати елемент
0
        self.list1.bind("<Double-Button-1>",
self.event2) # пов'язати подію з методом
        self.canvas1 = Canvas(root, width=200,
height=160, bg='white') # створити канву

self.line1=self.canvas1.create_line(0,0,100,100,
width=5) # створити лінію на канві
        self.canvas1.grid(row=2, column=0) #
розмістити
        self.canvas1.bind("<Motion>", self.event1) #
пов'язати подію з методом
        self.dv=DoubleVar() # створити дійсну змінну
        self.scale1 = Scale(self, from_=-10.0,
to=10.0, resolution=0.5, label='Scale',
orient=HORIZONTAL, variable=self.dv) # створити
```

```
шкалу, пов'язати зі змінною dv
        self.scale1.grid(row=2, column=1) #
розмістити
        menu1 = Menu(self) # створити меню
        master.config(menu=menu1) # установити меню
для вікна
        menu11 = Menu(menu1) # створити підменю
        menu1.add_cascade(label='Menu', menu=menu11)
# додати підменю

menu11.add_command(label='Exit',command=sys.exit) #
додати елемент меню
    def command1(self): # метод command1
        x=float(self.tv.get()) # присвоїти `x`
значення tv
        self.tv.set(x**2) # установити tv значення
x**2
        print self.bv.get() # вивести значення bv
        print self.iv.get() # вивести значення iv
        print self.dv.get() # вивести значення dv
        print
self.list1.get(self.list1.curselection()) # вивести
вибраний у списку елемент
    def event1(self,event): # метод event1 (обробник
події)
        self.tv.set(event.x) # установити tv значення
координати миші `x`
        self.canvas1.coords(self.line1,(event.x,
event.y, event.x+50, event.y+50)) # змінити
координати лінії line1
    def event2(self,event): # метод event2 (обробник
події)
        event.widget["fg"]="red" # змінити значення
властивості fg віджета, що викликав подію
        self.command1() # виклик методу command1
```

```
root = Tk() # створити головне вікно
app = MyFrame(master=root) # створити наш фрейм на
вікні
app.mainloop() # головний цикл обробки подій
root.destroy() # знищити вікно
```



Рисунок 9 - Вікно програми

ttk.Treeview - дерево елементів

Модуль `ttk` містить класи, які дозволяють використання віджетів `Tk` з підтримкою тем оформлення. Клас `ttk.Treeview` дозволяє відображати ієрархічну колекцію (дерево) елементів. Кожний елемент може мати текстовий надпис, рисунок і список значень даних.

```
import Tkinter, ttk
def btn1Click(event):
    '''Обробник події відпускання кнопки 1 миші'''
```

```
    tree = event.widget # віджет, що викликав подію
    node = tree.focus() # вибраний елемент дерева
(його id)
    print node # id
    print tree.item(node) # словник опцій вузла:
    #{'text':'', 'image':'', 'values':'', 'open':0,
'tags':''}
    print tree.item(node,'text') # текст вибраного
елемента
    # або print tree.item(node)['text']
    print tree.parent(node) # предок
    print tree.index(node) # індекс елемента в списку
споріднених
    print tree.prev(node) # попередній споріднений
    print tree.next(node) # наступний споріднений
    print tree.get_children(node) # список дочірніх
    print tree.set(node) # словник зі значеннями
колонок
    print tree.exists(node) # чи існує елемент?
def dbl_btn1Click(event):
    '''обробник події подвійного натиску кнопки 1
миші'''
    print 'dblClicked'
def btn3Click(event):
    '''обробник події натиску кнопки 3 миші'''
    print 'btn3Clicked'
def tagClicked(event):
    '''Обробник подій натиску мишею на тезі'''
    print 'tagClicked'
def treeOpenClose(event):
    '''Обробник подій відкриття і закриття
піддерева'''
    print 'opened/closed'
def treeSelect(event):
    '''Обробник події вибору елемента'''
```

```
    print 'selected'

root = Tkinter.Tk() # створити головне вікно
img = Tkinter.PhotoImage(file='folder.gif') # рисунок
sbar_y = ttk.Scrollbar(orient="vertical") # створити
вертикальну смугу прокручування
sbar_x = ttk.Scrollbar(orient="horizontal") #
створити горизонтальну смугу прокручування
tree = ttk.Treeview(height=10) # створити дерево
tree['selectmode']=Tkinter.EXTENDED # дозволити вибір
багатьох елементів
# або так:
tree.config(selectmode=Tkinter.NONE) # заборонити
вибір елементів
#tree.state(('disabled',)) # заблокувати tree
tree['columns'] = ('state',) # додати колонки
tree.column('state', width=100, anchor='center') #
параметри колонки 'state'
tree['displaycolumns']='state' # показувати колонку
tree.heading('#0', text='Item',image=img) # надпис на
колонці 0
tree.heading('state', text='State') # надпис на
колонці 'state'
sbar_y['command'] = tree.yview # під час
прокручування змінювати положення дерева
sbar_x['command'] = tree.xview
tree['yscrollcommand'] = sbar_y.set # значення
повзунка смуги прокручування
tree['xscrollcommand'] = sbar_x.set

# розмістити віджети
sbar_y.pack(side=Tkinter.RIGHT, fill=Tkinter.Y)
sbar_x.pack(side=Tkinter.BOTTOM, fill=Tkinter.X)
tree.pack(side=Tkinter.LEFT, fill=Tkinter.Y)
```

```
# прив'язки до обробників подій
tree.bind('<ButtonRelease-1>', btn1Click)
tree.bind('<Double-Button-1>', dbl_btn1Click)
tree.bind('<Button-3>', btn3Click)
tree.bind('<<TreeviewSelect>>', treeSelect) #
обробник події вибору
tree.bind('<<TreeviewOpen>>', treeOpenClose) #
обробник події відкриття піддерева
tree.bind('<<TreeviewClose>>', treeOpenClose) #
обробник події закриття піддерева

tree.insert('', 0, 'first', text='item 1', image=img)
# додати перший елемент 'first' після кореневого ''
tree.item('first', text='item 1!',open=1) # змінити
опції елемента 'first'
tree.set('first', 'state', '***') # значення для
'first' в колонці 'state'
id=tree.insert('', 'end', text='item 2') # додати
другий елемент після ''
id=tree.insert(id, 'end', text='item
21',tags=('tag1',)) # додати дочірні до id
tree.tag_configure('tag1', foreground='blue') # колір
тегу
tree.tag_bind('tag1', '<3>', tagClicked); # вказати
обробник події натиску на праву кнопку миші
tree.insert('first', 'end', 'child', text='Child') #
додати дочірній 'child' до 'first' в кінець
tree.insert('child', 'end', text='Child') # додати
дочірній до 'child'
tree.insert('first', 'end',
text='Child',values=('***',)) # додати дочірній до
'first'
tree.move('child', '', 'end') # перемістити 'child'
разом з дочірніми в кінець кореня
# або
```

```
#tree.detach('child') # відділити від дерева (зі
збереженням в пам'яті)
#tree.reattach('child', '', 'end') # знову прикріпити
до дерева (предок '', позиція 'end')
#tree.delete('child') # повністю видалити
#tree.set_children('child',id,'first') # замінює
дитину елемента 'child' новими дітьми (id,'first')
tree.focus('first') # установити фокус на перший
елемент
tree.selection_set(('first',)) # вибрати елементи
print tree.selection() # вибрані елементи
tree.see('first') # прокрутити дерево до елемента,
щоб він став у полі зору

root.mainloop()# головний цикл обробки подій
```

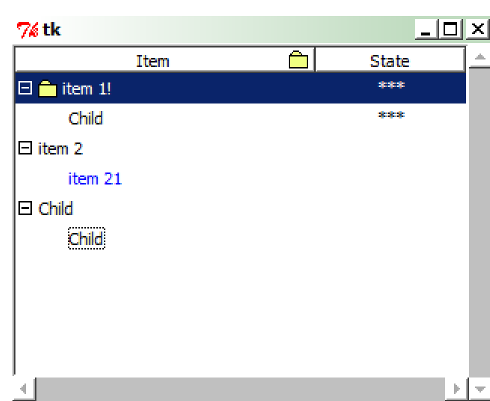


Рисунок 10 - Дерево елементів

Вбудовування інтерпретатора Python у C++ програму

Нижче показано приклад програми мовою C++, яка має можливість звернення до інтерпретатора Python. Якщо використовується середовище розробки Code::Blocks 16.01 та компілятор GNU GCC Compiler, то в опціях проекту (Project build

options) потрібно вказати шлях до заголовних файлів (Search directories) C:\Python27\include та під'єднати усі бібліотеки (Link libraries) з C:\Python27\libs. Якщо використовується середовище розробки Borland C++ Builder 6, то:

- Виконайте конвертацію бібліотеки: `coff2omf.exe python27.lib python27_.lib`.
- Скопіюйте python27_.lib в папку з проектом і переіменуйте його в python27.lib.
- Виберіть меню Project/Options…/Directories та додайте в Include path C:\Python27\include
- Додайте до проекту python27.lib та Python.h

```
#include "Python.h"
main(int argc, char **argv)
{
Py_SetProgramName(argv[0]); // передає argv[0]
інтерпретатору
Py_Initialize(); // ініціалізація інтерпретатора
// виконання команд Python (ніби модуль __main__)
PyRun_SimpleString("import time\n");
PyRun_SimpleString("print
time.localtime(time.time())\n");
Py_Finalize(); // закінчення роботи інтерпретатора
}
```

ctypes - виклик зовнішніх С-функцій

ctypes - це бібліотека для поступу до зовнішніх С-функцій, яка забезпечує сумісні з С типи даних і дозволяє виклик функцій з DLL або розподілюваних бібліотек Unix. Наступна С-функція `f` отримує три аргументи (змінну `n` та вказівники `x` і `A`) і повертає вказівник `B`. Зауважте, що функція змінює значення за адресами `x` і `A`.

```
#include <stdlib.h>
float* __declspec(dllexport) f(float* x, int n,
```

```
float* A)
{
float *B = (float*)malloc(sizeof(float) * n);
int i;
for(i=0; i<n; i++)
    {B[i]=A[i]+*x; A[i]-=*x;}
*x=A[1]-A[0];
return B;
}
```

Для компіляції цього коду в бібліотеку DLL застосовано команди GCC 4.9.2 (tdm-1):

```
mingw32-gcc.exe -O2 -c main.c -o main.o
mingw32-gcc.exe -shared main.o -o mydll.dll -s
```

Тепер до бібліотеки mydll.dll можна звернутись з Python:

```
from ctypes import *
mydll=cdll.LoadLibrary("mydll.dll") # завантажити
бібліотеку DLL
float3 = c_float * 3 # тип масиву з 3 елем. C-
сумісного типу float
A=float3(1, 2, 3) # масив
mydll.f.argtypes=[POINTER(c_float), c_int, float3] #
типи аргументів
mydll.f.restype=POINTER(c_float) # тип результату
x=c_float(1) # змінна C-сумісного типу float
B=mydll.f(x, 3, A) # виклик функції (x, A -
вказівники)
# або B=mydll.f(byref(x), 3, A)
for a in A: print a, # вивести масив A
print '\n', B[0], B[1], B[2] # вивести масив B (але
не B[3] !)
print x.value # значення змінної x
```

```
0.0 1.0 2.0
2.0 3.0 4.0
1.0
```

Розширення Python мовою C++

Нижче наведено послідовність дій для створення мовою C++ Python-модуля `Extest`, який містить функцію `fac`, що повертає факторіал числа. Створення модулів розширення мовою C++ дозволяє вирішити проблему низької продуктивності Python.

1.Вихідний код модуля розширення мовою C++ (main.cpp):

```
int fac(int n) // рекурсивна функція, повертає
факторіал
{
    if (n < 2) return(1);
    return (n)*fac(n-1);
}

#include "Python.h" // під’єднати файл Python.h
// функція повертає об’єкт Python типу int
static PyObject *Extest_fac(PyObject *self, PyObject
*args)
{
    int num;
    // конвертує дане Python типу int в C++ типу int
    if (!PyArg_ParseTuple(args, "i", &num))
        return NULL;
    // конвертує дане C++ типу int в Python типу int
    return (PyObject*)Py_BuildValue("i", fac(num));
}

// масив методів, які експортує модуль
static PyMethodDef ExtestMethods[] =
{{ "fac", Extest_fac, METH_VARARGS }, { NULL, NULL
},};
```

```
void initExtest() // функція ініціалізації модуля
{
    Py_InitModule("Extest", ExtestMethods);
}
```

2.Модуль Python, який створює і установлює модуль розширення за допомогою `distutils` (setup.py):

```
from distutils.core import setup, Extension
setup(name='Extest', ext_modules=[Extension('Extest',
sources=['main.cpp'])])
```

3.В командному рядку введіть (для Python 2.5 необхідне установлене MS Visual C++ 2003):

```
setup.py build
setup.py install
```

4.Перевірка роботи модуля в Python:

```
import Extest
Extest.fac(7) # 5040
```

РОЗДІЛ 2. СТОРОННІ БІБЛІОТЕКИ PYTHON

IPython - інтерактивна командна оболонка

IPython 5.X (https://ipython.readthedocs.io/en/5.x/) - це командна оболонка для інтерактивних обчислень на багатьох мовах програмування, яка забезпечує інтроспекцію, мультимедіа, доступ до системної оболонки, автодоповнення коду та історію команд. Початково розроблена для Python і є ядром більш масштабного проекту Jupyter. Широко використовується в екосистемі SciPy. Запускається командою `ipython` або `jupyter-qtconsole`.

Команди	Коментар
In [1]: x=1	Після запрошення `In [1]:` введіть потрібну команду.
In [2]: x=x+1;x=x+2	Або кілька команд.
In [3]: x **Out[3]:** 4	Вивести значення змінної.
In [4]: x;	Результат не виводити.
In [5]: _ **Out[5]:** 4	Змінна `_` містить попередній результат.
In [6]: In **Out[6]:** ['', u'x=1', u'x=x+1;x=x+2', u'x', u'x;', u'_', u'In']	Змінна `In` містить список комірок введення.
In [7]: Out **Out[7]:** {3: 4, 5: 4, 6: ['', u'x=1', u'x=x+1;x=x+2', u'x', u'x;', u'_', u'In', u'Out']}	Змінна `Out` містить словник комірок виведення.
In [8]: su<Tab> sum super	Автодоповнення клавішею `<Tab>`. Використовуйте також клавіші ↑ і ↓ для пошуку команд в історії та комбінацію `Ctrl-r` для відкриття вікна пошуку.
In [8]: sum?	Інформація про об’єкт.

Docstring: sum(iterable[, start]) -> value Return the sum of an iterable or sequence of numbers ...	
In [9]: sum??	Детальна інформація про об’єкт.
In [10]: !cd e:\Anaconda2\Scripts **In [11]:** !!cd **Out[11]:** ['e:\\Anaconda2\\Scripts']	Для доступу до команд системної оболонки використовуйте символи "!" або "!!".
In [12]: !cd <Tab>	Автодоповнення елементів каталогу клавішею <Tab>.
In [13]: files=!dir /B	Присвоїти змінній files список файлів поточного каталогу.
In [14]: file="main.py" **In [15]:** !dir $file	Передати значення змінної file команді dir.
In [16]: !dir {file}	Або так.
In [17]: %quickref	Магічна команда %quickref. Магічні команди починаються з "%" (рядкові команди) або "%%" (коміркові команди) і можуть мати аргументи.
In [18]: %lsmagic **Out[18]:** Available line magics: %alias %alias_magic %autocall ... Available cell magics: %%! %%HTML %%SVG %%bash ...	Список усіх магічних команд.
In [19]: %psearch s* set	Шукати усі об'єкти, що починаються з "s".

setattr ...	
In [20]: ?s*	Або так.
In [21]: %%writefile main.py ...: print sum(range(100)) ...: Writing main.py	Створити файл з наступним вмістом: print sum(range(100))
In [22]: %run main.py 4950	Виконати програму в IPython.
In [23]: %edit	Викликати зовнішній текстовий редактор і виконати введений код.
In [24]: %timeit sum(range(100)) 100000 loops, best of 3: 2.1 µs per loop **In [25]:** %time sum(range(100000)) Wall time: 8 ms **Out[25]:** 4999950000L	Визначити тривалість виконання коду.
In [26]: %history x=1 x=x+1;x=x+2 ...	Вивести усі введені команди.
In [27]: %history -f myhistory.py	Зберегти усі введені команди у файл.
In [28]: %pylab	Імпортувати модулі numpy та matplotlib.
In [29]: plt.plot(np.linspace(0,1)**2) **Out[29]:** [<matplotlib.lines.Line2D at 0x8cd2dd8>]	Побудувати графік функції $y=x^2$.

Jupyter Notebook - інтерактивні документи

Jupyter Notebook - це вільна веб-програма, яка дозволяє створювати і поширювати інтерактивні документи, які містять живий програмний код, формули, візуалізацію і форматований текст (https://jupyter.org). Notebook підтримує більше 40 мов програмування, у тому числі Python. Код може створювати мультимедійне інтерактивне виведення: HTML, рисунки, відео, LaTeX і довільні MIME-типи. Jupyter Notebook широко застосовується для інтерактивних обчислень в різних галузях науки і техніки. Форматований текст в Notebook можна створювати мовою розмітки Markdown (https://daringfireball.net/projects/markdown). Це мова розмітки, яка орієнтована на легкість створення і читання документу з подальшим його перетворенням у HTML. Зокрема, текст курсивом повинен знаходитись між символами `*`, жирний текст - між символами `**`, LaTeX формула - між символами `$` або `$$`, заголовок повинен починатись з символу `#` з наступним пробілом. Jupyter Notebook запускається командою `jupyter-notebook`. Нижче показані приклади документів Markdown та Jupyter Notebook.

```
# Заголовок
Markdown текст: *курсив* **жирний** `програмний код`
[Посилання](https://jupyter.org)
![рисунок](https://jupyter.org/assets/nav_logo.svg)

    >>> a=2 # програмний код
    >>> a**2

LaTeX формула: $a^{i\pi}_{k} + 1 =
\frac{\sqrt{x+1}}{\sin x}$

|A |B |C |
|--|--|--|
|1 |2 |3 |
```

Заголовок

Markdown текст: *курсив* **жирний** `програмний код` [Посилання]

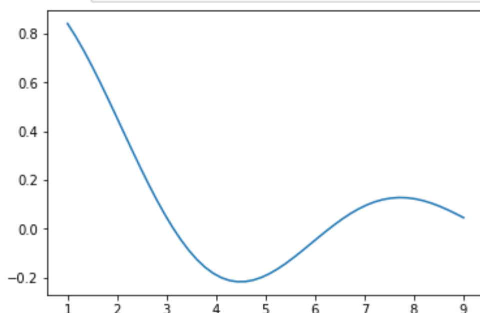
рисунок

```
>>> a=2 # програмний код
>>> a**2
```

LaTeX формула: $a_k^{i\pi} + 1 = \frac{\sqrt{x+1}}{\sin x}$

A	B	C
1	2	3

```
In [1]: %matplotlib notebook
        import numpy as np
        import matplotlib.pyplot as plt
        from ipywidgets import * # елементи керування
        x = np.linspace(1, 9); line, = plt.plot(x, np.sin(x)/x) # крива
        def update(A=(0,2,0.1), B="1.0", C=[0,1,2], D=False):
            line.set_ydata((-1 if D else 1)*A*np.sin(float(B)*x+C)/x)
            plt.show() # оновити графік
        interact(update); # інтерактивний режим
```



A 1.00

B 1.0

C 0

D

```
In [2]: print "неформатований текст"
        from IPython.display import display, HTML, Markdown, SVG
        # візуалізація коду мовами HTML, Markdown, SVG
        display(HTML(u"HTML <b>жирний</b> текст"))
        display(Markdown(u"Markdown **жирний** текст"))
        display(SVG("""<svg xmlns="http://www.w3.org/2000/svg" width="
        <circle r="10" cx="50%" cy="50%" fill="green"/></svg>"""))
```

```
неформатований текст
```

HTML **жирний** текст

Markdown **жирний** текст



Рисунок 11 - Вигляд документа Jupyter Notebook

Matplotlib - процедурний API pyplot

Matplotlib (http://matplotlib.org) є бібліотекою для побудови різноманітних 2D діаграм у різних форматах і для різних інтерактивних середовищ. `matplotlib.pyplot` – це її простий у використанні інтерфейс у стилі MATLAB. Нижче показано приклад створення графіка з лінією між точками (0,0) і (1,1) за допомогою Matplotlib 2.1.1.

```
import matplotlib.pyplot as plt # імпортувати модуль matplotlib.pyplot як plt
plt.plot([0,1],[0,1], 'o-k') # створити лінію
plt.show() # показати рисунок
```

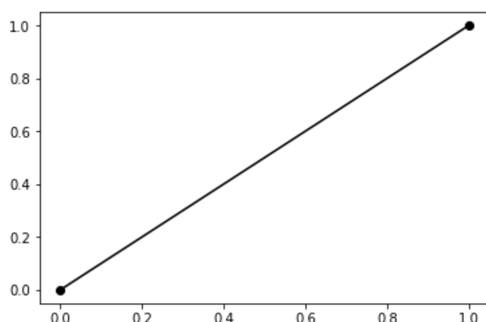


Рисунок 12 - Приклад використання matplotlib.pyplot

Matplotlib - об’єктно-орієнтований API

Об’єктно-орієнтований інтерфейс програмування Matplotlib використовує об’єкти (таких класів як `figure.Figure`, `axes._subplots.AxesSubplot`, `lines.Line2D`) і їх методи для побудови графіків. Складніший у використанні ніж процедурний інтерфейс pyplot, але має більше можливостей для налаштування графіків.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots() # створити об'єкти рисунка і системи координат
#fig, ax = plt.figure(), plt.axes(yscale='log') # або з логарифмічною шкалою 0y
line,=ax.plot([0,2],[0,1]) # створити лінію в системі координат
line.set_linewidth(2) # ширина лінії
line.set_color('r') # колір лінії (або 'red', або (1.0,0.2,0.3), або '0.7')
line.set_linestyle('--') # стиль лінії (або '-', '-.', ':', )
line.set_marker('s') # маркери точок (або один з символів .,ov^<>1234sp*hH+xDd|_)
```

```
line.set_markersize(10) # розміри точок
ax.axis('equal') # однаковий масштаб осей
plt.show() # показати рисунок (або fig.show())
```

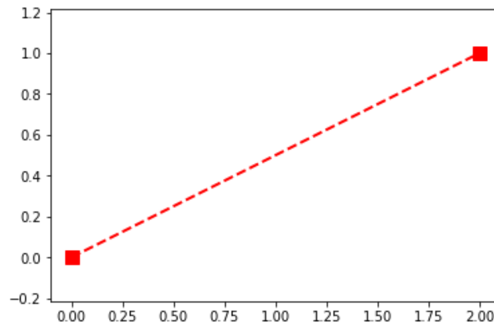


Рисунок 13 - Приклад використання об’єктно-орієнтованого API Matplotlib

Matplotlib - додаткові параметри графіків

В прикладі показно створення вкладених графіків (subplot) і використання їх додаткових параметрів, таких як назви і масштаб осей, заголовок, сітка, надписи. Можливо налаштувати деякі параметри за замовчування в словнику `plt.rcParams`.

```
import numpy as np # імпортувати модуль numpy як np
import matplotlib.pyplot as plt # імпортувати модуль
matplotlib.pyplot як plt
# параметри графіків за замовчуванням:
plt.rcParams['lines.color'] = 'k'
#plt.rcParams['image.aspect'] = 'equal' # але це не
працює у версії 2.1.1
#plt.rcParams.keys() # список усіх параметрів

f=lambda x: np.cos(np.pi*x)*x # функція повертає
x*cos(pi*x)
```

```
x1 = np.arange(0.0, 5.0, 0.1) # масив з прогресії
x2= np.arange(0.0, 5.0, 0.02) # масив з прогресії
plt.subplot(2,1,1) #діаграма 1 (рядків 2, колонок 1,
номер 1)
plt.plot(x1,f(x1),'bo',x2,f(x2),'k') # криві
plt.title('Figure1') # заголовок
plt.subplot(2,1,2) # діаграма 2 (рядків 2, колонок 1,
номер 2)
line1,line2=plt.plot([1,2,3],[2,4,8],'r-
',[1,2,3],[1,2,1],'b-') # криві line1,line2
line3,=plt.plot([1,2,3],[1.5,3,4.5],'g--') # крива
line3
plt.setp((line1,line2),linewidth=2.0) # властивості
кривих line1,line2
plt.title('Figure2') # заголовок
plt.xlabel('x'); plt.ylabel('y') # надпис осей x i y
plt.text(2, 2, r'$\sigma=2/6$') # LaTeX текст на
діаграмі
plt.axis([1, 3, 0, 10]) # розміри осей x,y
# plt.axis('equal') # однаковий масштаб осей
plt.grid(True) # сітка
plt.show() # показати графік
```

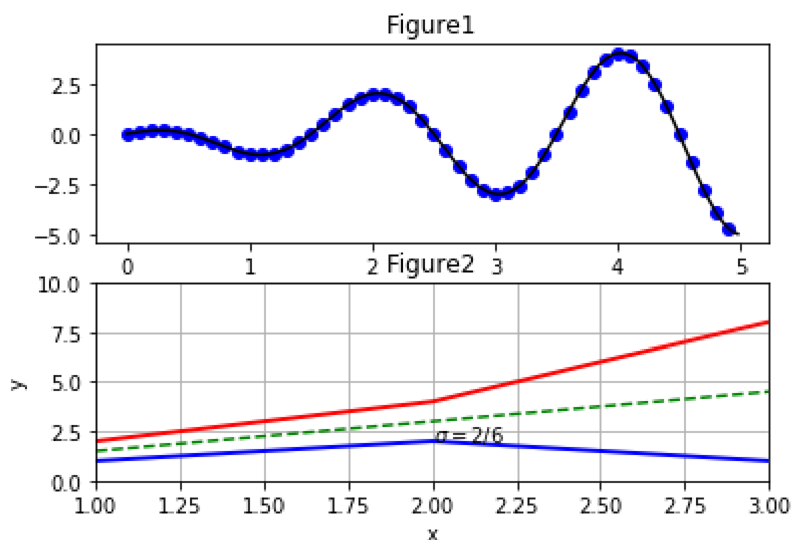


Рисунок 14 - Створення вкладених графіків

Matplotlib - інші типи діаграм

В прикладі показано створення діаграм розсіювання, гістограм, контурних діаграм та тривимірних графіків. Інші приклади використання Matplotlib для створення діаграм різного типу можна подивитись тут (http://matplotlib.org/gallery/index.html#).

```
import numpy as np
import matplotlib.pyplot as plt

# діаграма розсіювання
x=[0, 1, 2, 1] # координати точок
y=[0, 1, 4, 5]
colors=[0.1, 0.4, 0.7, 0.8] # колір точок
sizes=[20,40,60,80] # розміри точок
plt.scatter(x, y, c=colors, s=sizes, alpha=0.7)
plt.xlabel('x');plt.ylabel('y');plt.grid();plt.show()
```

```
print "Рисунок - Діаграма розсіювання"

# гістограми
x1=np.random.normal(0, 1.0, 100) # випадкова величина
x2=np.random.normal(2, 1.0, 100) # випадкова величина
plt.figure()
plt.hist(x1, alpha=0.5, bins=7) # гістограма
plt.hist(x2, alpha=0.5, bins=7) # гістограма
plt.xlabel('x');plt.ylabel('y');plt.grid();plt.show()
print "Рисунок - Гістограми"

# контурна діаграма для даних X, Y, Z
X, Y = np.meshgrid(np.linspace(0, 9), np.linspace(0,
9))
Z = X**2+Y**2
plt.figure()
plt.contour(X, Y, Z, 5, colors='white') # без
заповнення
plt.contourf(X, Y, Z, 5, cmap=plt.cm.gray) # з
заповненням
# або відображення зображень
#plt.imshow(Z, extent=[0, 9, 0, 9], origin='lower',
cmap=plt.cm.gray)
#plt.axis(aspect='image') # пропорції осей
plt.colorbar() # смуга зі значеннями Z
plt.xlabel('x');plt.ylabel('y');plt.show()
print "Рисунок - Контурна діаграма"

# тривимірні графіки
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure() # рисунок
ax = Axes3D(fig) # система координат
ax.scatter3D([0,10],[0,10],[0,200], s=200) # точки
ax.plot3D([10,0],[0,10],[0,200], 'ko:') # лінії
#ax.plot_wireframe(X, Y, Z) # каркасна поверхня
```

```
ax.plot_surface(X, Y, Z) # поверхня
ax.set_xlabel('x');ax.set_ylabel('y');ax.set_zlabel('
z');plt.show()
print "Рисунок - Тривимірний графік"
```

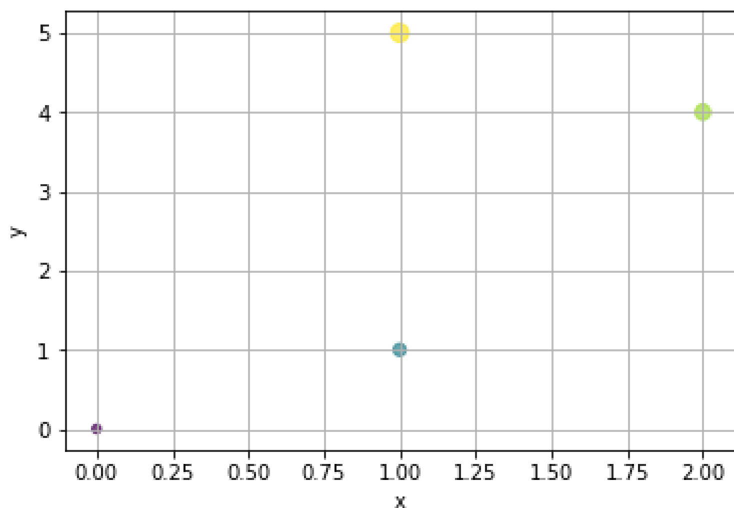


Рисунок 15 - Діаграма розсіювання

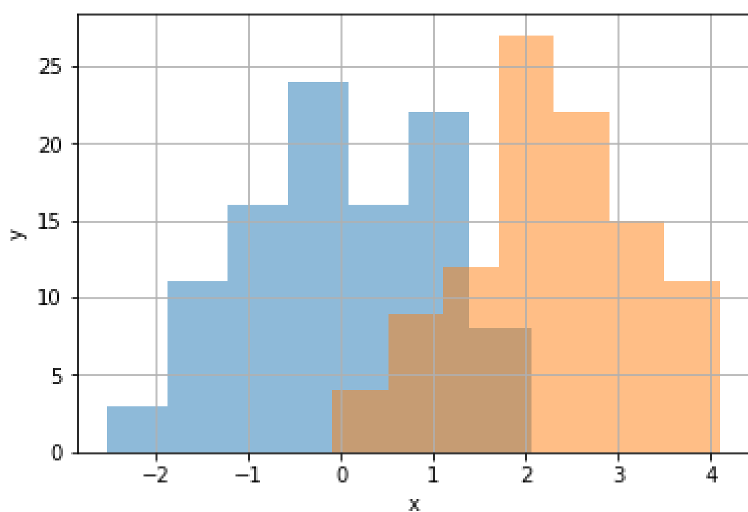


Рисунок 16 - Гістограми

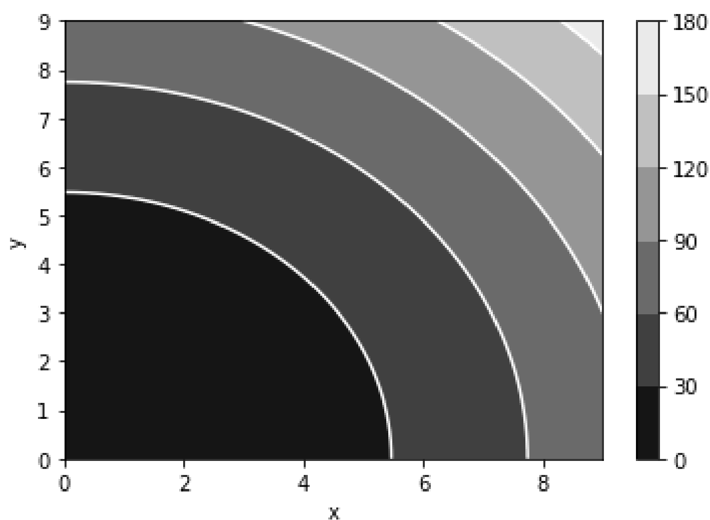


Рисунок 17 - Контурна діаграма

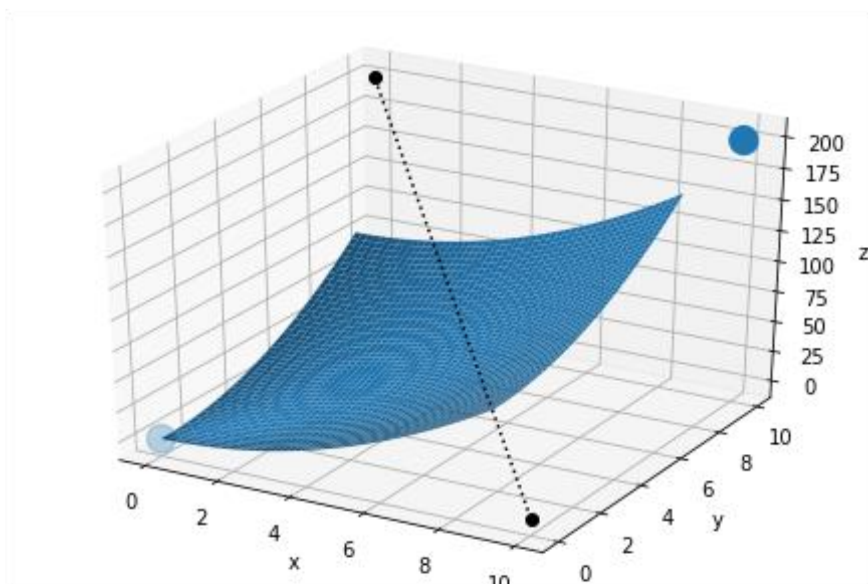


Рисунок 18 - Тривимірний графік

Matplotlib - інтерактивна побудова графіків

В прикладі графік інтерактивно перебудовується під час натиску клавіш “стрілка вгору” і “стрілка вниз”. Для цього подія `key_press_event` пов’язується з функцією обробки події `keyPress`. Цю програму бажано виконувати так: python.exe main.py

```
import matplotlib.pyplot as plt
from random import randint

def keyPress(event): # функція обробки подій
    if event.key=='up': # якщо натиснута стрілка
вгору
        X.append(randint(0,10)) # додати в список X
випадкове число
        Y.append(randint(0,10)) # додати в список Y
```

```
випадкове число
    if event.key=='down': # якщо натиснута стрілка
вниз
        if X: X.pop() # вилучити зі списку X останнє
число
        if Y: Y.pop() # вилучити зі списку Y останнє
число
    ln.set_data(X,Y) # установити дані для полілінії
    plt.draw() #ln.figure.canvas.draw() #
перерисувати

X=[] # список координат x
Y=[] # список координат y
ln,=plt.plot(X, Y, 'k-o') # полілінія
plt.gcf().canvas.mpl_connect('key_press_event',
keyPress) # пов'язати подію натиску клавіш з функцією
обробки подій
plt.axis([0, 10, 0, 10]) # шкала осей
plt.xlabel("x");plt.ylabel("y");plt.show()
print "Рисунок - Інтерактивний графік"
```

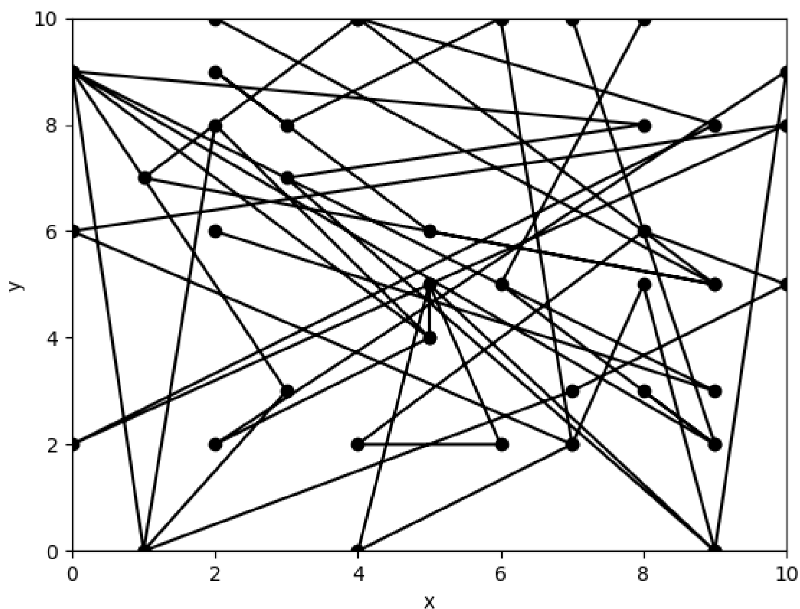


Рисунок 19 - Інтерактивний графік

Bokeh - інтерактивна візуалізація

Bokeh 0.13 (http://bokeh.pydata.org) - це бібліотека для інтерактивної і високопродуктивної візуалізації в сучасних браузерах. Використовується для створення інтерактивних програм для візуалізації даних. Цей приклад створює графік, який розташований в незалежному html-документі з javascript-сценаріями.

```
import numpy as np
from bokeh.plotting import figure, show, output_file
x = np.linspace(-1,1,1000) # дані для візуалізації
y = np.sin(1/x)
dy= np.random.normal(0, 0.2, len(x))
```

```
output_file("plot.html") # документ для виведення
plot = figure(plot_height=200,
output_backend="webgl") # рисунок
plot.line(x, y, line_width=3) # крива
plot.circle(x[::100], y[::100], fill_color="white",
line_color="red", size=10) # точки на кривій
plot.scatter(x, y+dy, alpha=0.5) # випадкові точки
show(plot) # показати рисунок в браузері
```

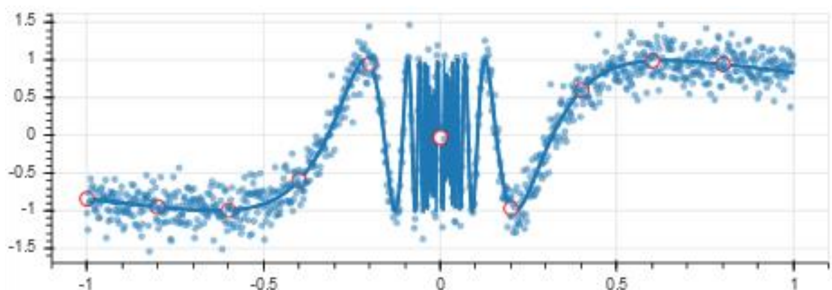


Рисунок 20 - Вигляд графіка в браузері

Bokeh - серверна програма

За допомогою сервера застосувань Bokeh можуть бути створені клієнтські html-документи, які взаємодіють з серверною Python-програмою. Для виконання прикладу введіть в консолі:

```
e:/anaconda2/scripts/bokeh serve --show main.py
```

```
import numpy as np
from bokeh.io import curdoc
from bokeh.layouts import row, widgetbox
from bokeh.models import ColumnDataSource
from bokeh.models.widgets import Slider
from bokeh.plotting import figure
def update(attrname, old, new): # викликається під
час прокручування
    y=np.sin(slider.value*x) # нові значення
```

```
    source.data=dict(x=x, y=y) # установити нові дані
N = 100 # кількість точок
x = np.linspace(0, 4*np.pi, N)
y = np.sin(x)
source = ColumnDataSource(data=dict(x=x, y=y)) #
початкові дані
plot = figure(plot_height=200, plot_width=400,
x_range=[0, 4*np.pi], y_range=[-2, 2],
output_backend="webgl") # рисунок
plot.line('x', 'y', source=source, line_width=3) #
крива
slider = Slider(title="частота", value=1.0,
start=0.1, end=3.0, step=0.1) # віджет повзунок
slider.on_change('value', update) # пов'язати подію з
функцією
wb = widgetbox(slider) # контейнер з віджетом
curdoc().add_root(row(wb, plot)) # розмістити на
документі в ряд
```

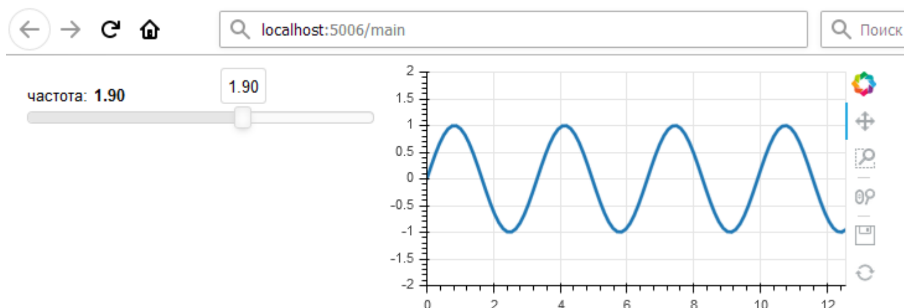


Рисунок 21 - GUI програми в браузері

numpy - робота з масивами

NumPy (http://www.numpy.org, http://scipy.org) – вільна бібліотека Python для високопродуктивних операцій з багатовимірними масивами (у тому числі матрицями). NumPy є основою таких бібліотек для роботи з даними як SciPy, Matplotlib,

pandas, scikit-learn та багатьох інших. Часто застосовується разом з бібліотекою SciPy, яка містить багато зручних і ефективних чисельних процедур (для інтегрування, оптимізації, інтерполяції, статистики, обробки сигналів та іншого) [14, 31]. NumPy та SciPy можна розглядати як вільну альтернативу MATLAB. В прикладах використовується NumPy 1.13.3 та SciPy 0.19.1. В цьому прикладі показані базові операції з масивами: створення, властивості, доступ до частин масиву (зрізи), зміна форми, арифметичні операції, математичні функції, способи індексації, збереження у файлах, створення масивів з різнотипними елементами.

```
import numpy as np
#print numpy.lookfor("create array") # шукати "create
array" серед документації

# створення масивів:
np.array([1.0,2.0,3.0,4.0]) # одновимірний масив
np.array([1,2,3,4], dtype=int) # одновимірний масив
цілих чисел
np.array([[1.0,2],[3,4]]) # двовимірний масив дійсних
чисел
print np.array(range(6)) # одновимірний масив з
прогресії
np.arange(6) # або так
print np.linspace(start=0,stop=10,num=5) # масив з
рівномірно розподіленими значеннями
np.zeros((2,2)) # двовимірний нульовий масив
np.ones(2) # одновимірний масив з одиниць
np.full(2, 1) # або так
np.identity(2) # одинична матриця
np.random.random(5) # масив з випадковими значеннями
np.random.normal(loc=5, scale=1, size=5) # масив з
випадковими значеннями (нормальний закон)

a=np.array([[1,2,3], [4,5,6]])
```

```
a.ndim # кількість вимірів масиву
a.shape # розмір кожного виміру
a.size # загальний розмір
a.dtype # тип даних
a.tolist() # перетворити у список

# зрізи над масивом у форматі: a[початок:кінець:крок]
print a[0,1], a[0][1], a[0], a[-1], a[:,0], 
a[0:2:2,0:3:2]
a[0,0]=1.2 #змінити елемент з індексами 0,0
#Увага! a[0,0]==1 бо масив цілого типу
a[0]=np.array([1,2,3]) #змінити рядок з індексом 0
a0=a[0] # це не окрема копія першого рядка масиву a
a0[0]=2 #Увага! Масив a зміниться!
a0=a[0].copy() # це окрема копія першого рядка масиву 
a

print a.reshape((3,2)) # повертає масив зі зміненою 
формою
a.shape=(3,2) # або змінити форму масиву
a.resize((3,3)) # змінити форму масиву і заповнити 
нові комірки нулями
a.transpose() # транспонувати (або a.T)
np.ones(2)[:, np.newaxis] # перетворити в вектор-
стовпчик
np.ones(2).reshape((2,1)) # або так

np.concatenate([a, a]) # об'єднати масиви по 
вертикалі
np.vstack([a, a]) # або так
np.concatenate([a, a],axis=1) # об'єднати масиви по 
горизонталі
np.hstack([a, a]) # або так
np.split(a, [1]) # розбити масиви по вертикалі
#див. також np.vsplit, np.hsplit
```

```
# арифметичні операції над масивами
a+1 # додати 1 до кожного елемента
np.sqrt(a+1) # застосування математичних функцій
a+a # поелементне додавання
print np.array([1,2])+np.array([[1,2],[3,4]]) #
додавання масивів різного розміру

np.sum(a) # сума елементів
np.sum(a, axis=0) # суми в стовпцях
np.add.reduce(a, axis=0) # або так
np.cumsum(a) # накопичувальна сума
np.mean(a) # середнє
np.std(a) # стандартне відхилення
np.min(a), np.max(a) # мінімальне, максимальне
np.argmin(a) # індекс найменшого елемента
np.sort(np.array([3,2,7,1])) # сортувати
print np.argsort(np.array([3,2,7,1])) # індекси для
сортування

a=np.array([1,2,3,4])
a[[1,2]] # масив елементів з індексами 1,2
a[np.array([1,2])] # або так
a[[1,2]]=[2,3] # можна також змінювати масив `a`
a[np.array([False, True, True, False])] # або так
np.array([[1,2],[4,5]])[1,[0,1]] # комбінована
індексація

b=a<4 # масив з результатами логічного виразу
(dtype=bool)
b=(a > 2) & (a < 4) #або складні логічні вирази
a[b] # масив елементів з індексами b
print np.where(a<4) # масив індексів, де виконується
умова
np.any(a<4) # чи будь-який елемент
```

```
np.all(a<4) # чи усі елементи

# збереження у файлах
#a.tofile("myfile") # зберегти у файл
#a=np.fromfile("myfile", dtype=int) # прочитати з
файлу
# або
#np.save("myfile.npy",a) # зберегти у файл
#a=np.load("myfile.npy") # прочитати з файлу

a = np.zeros(2, dtype=('i4,f4,a10')) # масив з
різними типами
a[0]=(1, 10.0, 'A') # перший елемент
a[1]=(2, 20.0, 'B') # другий елемент
# або
a = np.zeros(2,
dtype={'names':('i','x','name'),'formats':('i4','f4',
'a10')})
a[0]=(1, 10.0, 'A') # перший елемент
a[1]=(2, 20.0, 'B') # другий елемент
a['name'] # стовпчик 'name'
a['name']=['a','b'] # змінити значення стовпчика
print a
print a[a['x'] < 20] # ті елементи, де x<20
```

```
[0 1 2 3 4 5]
[  0.    2.5   5.    7.5  10. ]
2 2 [1 2 3] [4 5 6] [1 4] [[1 3]]
[[2 2]
 [3 4]
 [5 6]]
[[2 4]
 [4 6]]
[3 1 0 2]
(array([0, 1, 2], dtype=int64),)
```

```
[(1,  10., 'a') (2,  20., 'b')]
[(1,  10., 'a')]
```

numpy.linalg - лінійна алгебра

Модуль містить базові інструменти лінійної алгебри: для декомпозиції матриць, розрахунку власних значень, визначника, норми матриці, розв’язування систем лінійних рівнянь та інвертування матриць. В прикладі також показано відмінність типів matrix і ndarray. Модуль `scipy.linalg` містить функції `numpy.linalg` та деякі додаткові функції, але може бути швидшим.

```
import numpy as np
A = np.matrix([[3, 1], [1, 2]]) # матриця
print A*A # множення матриць

print np.linalg.det(A) # визначник матриці (якщо не
0, то існує обернена матриця до A)

W,V=np.linalg.eig(A) # власні значення і власні
вектори
print W[0],V[:,0] # перше власне значення і
відповідний власний вектор
print A*V[:,0] - W[0]*V[:,0] # перевірка
A*V[:,i]=W[i]*V[:,i]

# розв'язування систем лінійних рівнянь AX=B (A,B -
матриці)
A = np.matrix([[3, 1], [1, 2]]) # матриця
B = np.matrix([[9], [8]]) # матриця
X = np.linalg.solve(A, B) # розв'язати систему
# X = A**(-1)*B # або шляхом інвертування матриці A
# A*X-B # перевірка (нульова матриця)
print X
```

```
# розв'язування систем лінійних рівнянь AX=B (A,B - масиви)
A = np.array([[3, 1], [1, 2]]) # масив
B = np.array([9, 8]) # масив
X = np.linalg.solve(A, B) # розв'язати систему
# X = np.linalg.inv(A).dot(B) # або шляхом інвертування матриці A
# np.dot(A, X) - B # перевірка (нульовий масив)
```

```
[[10  5]
 [ 5  5]]
5.0
3.61803398875 [[ 0.85065081]
 [ 0.52573111]]
[[  0.00000000e+00]
 [  2.22044605e-16]]
[[ 2.]
 [ 3.]]
```

numpy.random - генератори випадкових чисел

Модуль `numpy.random` містить функції для генерації випадкових чисел з різними розподілами ймовірностей.

```
import numpy as np
print np.random.random(3) # випадкова вибірка з інтервалу [0.0, 1.0) рівноімовірного розподілу
X=np.random.uniform(10,20,1000) # випадкова вибірка з рівноімовірного розподілу (ліва границя 10, права границя 20)
print X.mean(), X.std(), X.var() # середнє, середньоквадратичне відхилення, дисперсія
#або np.mean(X), np.std(X), np.var(X)
X=np.random.triangular(10,15,20,1000) # випадкова вибірка з трикутного розподілу (ліва границя 10, середнє 15, права границя 20)
```

```
print X.mean(), X.std(), X.var()
X=np.random.normal(15,1,1000) # випадкова вибірка з нормального розподілу (середнє 15, середньоквадратичне відхилення 1)
print X.mean(), X.std(), X.var()
```

```
[ 0.8374071   0.30127402  0.74989105]
15.101728332 2.93002837893 8.58506630134
14.9962835669 2.08557855706 4.34963791765
14.9576149223 0.973514441406 0.947730367627
```

numpy - поліноми

В прикладі показано роботу з поліномами в NumPy: створення, отримання коренів, апроксимація поліномом.

```
import numpy as np
p = np.poly1d([3, 2, -1]) # поліном 3*x**2 + 2*x - 1
print p(10) # значення полінома для x=10
print p.roots # корені
x, y = np.array([0,1,2]), np.array([0,2,8]) # дані
print np.polyfit(x, y, 2) # коеф. полінома 2 степеня, що апроксимує ці дані
```

```
319
[-1.          0.33333333]
[  2.00000000e+00  -1.18450880e-15   1.17986445e-16]
```

scipy.vectorize - векторизація функцій

Функція `scipy.vectorize` визначає векторизовану функцію, яка отримує послідовність або масив numpy і повертає один або кортеж масивів numpy. Використовується для перетворення звичайних функцій в їх векторизований варіант.

```
from scipy import vectorize
def f(x): return x+2 # звичайна функція
#print f([1,2,3]) # буде помилка!
fv=vectorize(f) # векторизована функція (приймає та повертає вектори)
print fv([1,2,3]) # повертає масив ndarray
```

```
[3 4 5]
```

scipy - похідна і первісна функції

В прикладі дано функцію $y = x^2 + 4x$. Шляхом чисельного диференціювання знаходиться її похідна $\dot{y} = dy/dx = 2x + 4$. Шляхом кумулятивного інтегрування знаходиться первісна $Y = \int y dx = x^3/3 + 2x^2 + const$ (дивись документацію функції `scipy.integrate.cumtrapz`).

Похідна і первісна можуть бути використані для пошуку екстремумів функції та для розв'язування рівнянь. Для пошуку екстремуму y потрібно розв'язати рівняння $dy/dx = 0$. А для пошуку кореня потрібно шукати екстремум первісної Y.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import cumtrapz

X=np.linspace(-5,3)
Y=X**2+4*X # функція
plt.plot(X,Y,'k-')

Y1=np.diff(Y)/np.diff(X) # похідна
plt.plot(X[:-1],Y1,'k--')
Y1=np.gradient(Y,X[1]-X[0]) # або
plt.plot(X,Y1,'k--')

Y_int=cumtrapz(Y, X, initial=0) # первісна
plt.plot(X,Y_int,'k:')
```

```
plt.xlabel('$x$');plt.ylabel('$y, \dot{y}, Y$');plt.grid();plt.show()
```

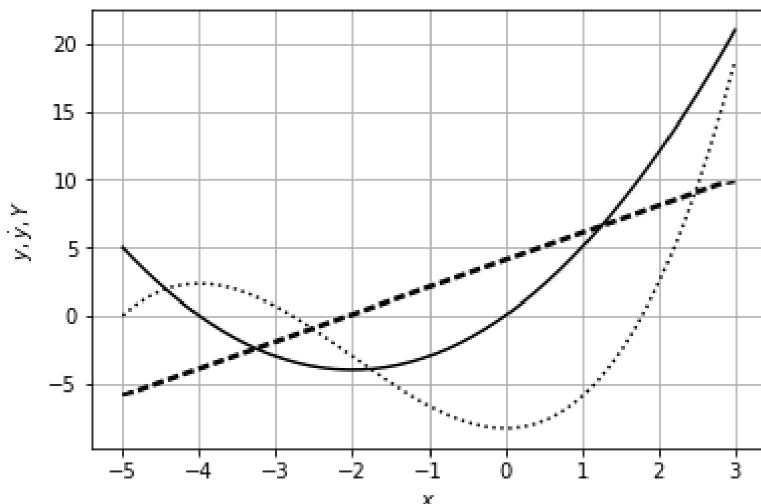


Рисунок 22 - Функція $y = x^2 + 4x$ (-), її похідна (--) і первісна (..)

scipy.integrate - інтегрування

Модуль `scipy.integrate` містить функції для інтегрування, у тому числі для інтегрування звичайних диференціальних рівнянь. В прикладі дано функцію $y = x^2$. Розраховується визначений інтеграл $\int_{-3}^{3} y dx$.

```
import numpy as np
from scipy.integrate import quad
f = lambda x,a: x**a # функція
print quad(f, -3, 3, args=(2,)) # результат інтегрування і оцінка абсолютної похибки результату
x=np.array([-3,-2,-1,0,1,2,3])
print np.trapz(f(x,2),x) # інтегрувати задану масивом функцію методом трапецій
```

```
(18.0, 1.9984014443252818e-13)
19.0
```

scipy.integrate.odeint - звичайні диференціальні рівняння

Функція `scipy.integrate.odeint` розв'язує систему звичайних диференціальних рівнянь з початковою умовою. В прикладі розв'язується просте диференціальне рівняння з початковою умовою $y(0) = 0$:

$$\frac{dy}{dt} = t^2.$$

```
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

def deriv(y,t): # функція повертає похідну dy/dt в
точці t
    return t**2 # значення правої частини рівняння
t = np.linspace(0, 1, 100) # час
y = odeint(deriv, y0=0, t=t) # інтегрує диф. рівняння
plt.plot(t,y,'k-') # рисує залежність y(t)
plt.xlabel('t');plt.ylabel('y');plt.grid();plt.show()
```

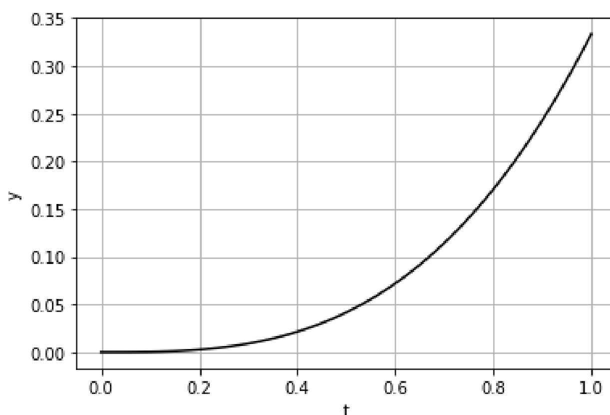


Рисунок 23 - Розв’язок диференціального рівняння

scipy.integrate.odeint - модель польоту снаряду

Модель польоту снаряду, випущеного під кутом 45 градусів до горизонту. Систему диференціальних рівнянь другого порядку

$$d^2x/dt^2 = 0,$$

$$d^2y/dt^2 = -9.8$$

перетворимо в систему диференціальних рівнянь першого порядку

$$dx'/dt = 0,$$

$$dx/dt = x',$$

$$dy'/dt = -9.8,$$

$$dy/dt = y',$$

де x, y - переміщення; *x’*, *y’* - швидкості. Початкові умови: *x*=0, *x’*=50, *y*=0, *y’*=50. Знайти функції *x*, *x’*, *y*, *y’*.

```
from scipy.integrate import odeint
import numpy as np
```

```
import matplotlib.pyplot as plt
def deriv(xy,t):
    x=xy[0] # переміщення по x
    x_=xy[1] # швидкість по x
    y=xy[2] # переміщення по y
    y_=xy[3] # швидкість по y
    return np.array([x_, 0.0, y_, -9.8]) # повертає значення функцій dx/dt, dx'/dt, dy/dt, dy'/dt
t = np.linspace(0.0, 10.0, 100) # час
init = np.array([0.0, 50.0, 0.0, 50.0]) # початкові умови для x, x', y, y'
xy = odeint(deriv, init, t) # інтегруємо систему диф. рівнянь, отримуємо масив [x, x', y, y']
plt.plot(xy[:,0], xy[:,2]) # траєкторія
plt.xlabel("x");plt.ylabel("y");plt.grid();plt.show()
```

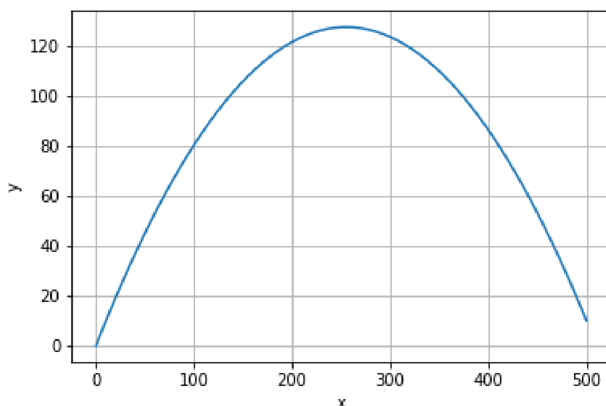


Рисунок 24 - Траєкторія переміщення y(x)

scipy.integrate.odeint - модель коливань, що згасають

Модель коливань пружини з масою *m*, жорсткістю *j* та коефіцієнтом демпфування *c* описується диференціальним рівнянням другого порядку

$$m\frac{d^2y}{dt^2} + c\frac{dy}{dt} + jy = 0.$$

Спочатку його необхідно перетворити в систему диференціальних рівнянь першого порядку

$$\frac{dy}{dt} = y',$$

$$\frac{dy'}{dt} = \frac{-jy - cy'}{m};$$

де *y* - переміщення, *y'* - швидкість, *dy'/dt* - прискорення. Початкові умови: y=1, y'=0. Знайти функції *y* та *y'*.

```
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

def deriv(y,t):
    m=1.0; j=1.0; c=0.1
    return [y[1], (-j*y[0]-c*y[1])/m] # повертає
значення функцій dy/dt та dy'/dt
t = np.linspace(0.0, 10.0, 100) # час
yinit = np.array([1.0, 0.0]) # початкові умови для
`y` та `y'`
y = odeint(deriv, yinit, t) # інтегруємо систему диф.
рівнянь, повертає двовимірний масив зі значеннями `y`
та `y'`
plt.plot(t, y[:,0], 'k-', t, y[:,1], 'k--') #
переміщення і швидкість
plt.xlabel("t");plt.ylabel("y,
y'");plt.grid();plt.show()
```

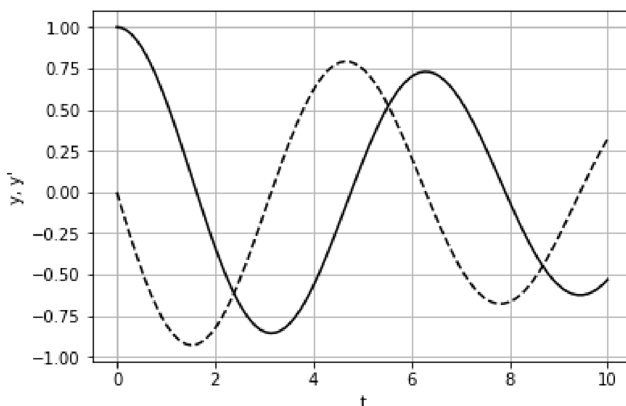


Рисунок 25 - Функції переміщення *y* (-) і швидкості *y'* (–)

scipy.interpolate - інтерполяція

Інтерполяція - це спосіб знаходження проміжних значень величини за її відомим дискретним набором значень. Для інтерполяції і апроксимації сплайнами застосовують функції з модуля `scipy.interpolate` (`interp1d`, `UnivariateSpline`, `interp2d`, `SmoothBivariateSpline` та інші). Сплайн - це функція, область визначення якої розбита на частини, на кожній з яких функція є певним поліномом.

```
import numpy as np
from scipy.interpolate import interp1d,
UnivariateSpline, interp2d
f=lambda x: x**x # функція
x = np.arange(4)
y = f(x)
print x,y # дискретний набір значень
y1 = interp1d(x, y, kind='linear') # лінійна
інтерполяція
y2 = interp1d(x, y, kind='quadratic') # інтерполяція
квадратичним сплайном
```

```
y2 = UnivariateSpline(x, y, k=2, s=0) # або (s -
коеф. згладжування)
kn=y2.get_knots() # вузли сплайна
print y1(2.5), y2(2.5), f(2.5) # інтерпольовані і
дійсне значення в точці x=2.5

import matplotlib.pyplot as plt
x=np.linspace(0,x.max(),100)
plt.plot(x,f(x),'k-',x,y1(x),'k--',x,y2(x),'k:') #
графіки
plt.scatter(kn,y2(kn)) # вузли кв. сплайна
plt.xlabel('x'),plt.ylabel('y'),plt.grid(),plt.show()

f=lambda x,y: x**2+y**2 # функція двох змінних
x=np.array([0,1,2])
y=np.array([0,1,2])
xx, yy = np.meshgrid(x,y) # сітка
z=f(xx,yy) # значення функції у вузлах сітки
z1 = interp2d(x, y, z, kind='linear') # двовимірна
лінійна інтерполяція даних x,y,z
print z1(0.5,0.5), f(0.5,0.5)  # інтерпольоване і
дійсне значення в точці 0.5, 0.5
```

```
[0 1 2 3] [ 1  1  4 27]
15.5 12.6458333333 9.88211768803
```

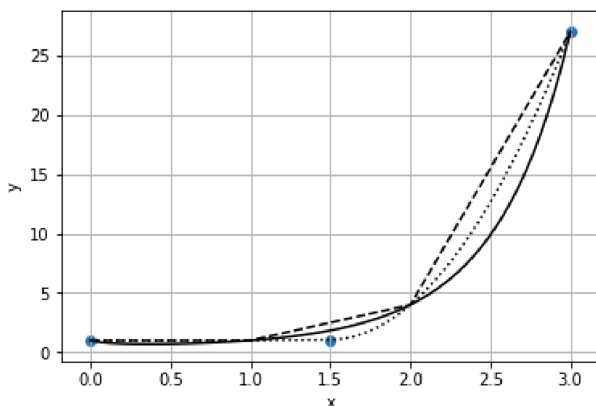


Рисунок 26 – Функція (-) та її лінійна (--) і квадратична (..) інтерполяції сплайнами

```
[ 1.] 0.5
```

scipy.optimize.fsolve - розв’язування рівнянь

Для розв’язування нелінійного рівняння чисельним методом застосовують функцію `scipy.optimize.fsolve`. Для розв’язування систем нелінійних рівнянь див. `scipy.optimize.root`. В прикладі розв’язується рівняння $x^2 - 2 = 0$.

```
import numpy as np
from scipy.optimize import fsolve
def f(x,a): # функція (x - вектор, a - константа)
    return x**2-a # повертає ліву частину рівняння
x0 = fsolve(f, np.array([-10,10]), args=(2,)) #
розв'язати рівняння з початковими значеннями коренів
x0=[-10,10]
print x0 # корені рівняння
```

```
[-1.41421356  1.41421356]
```

scipy.optimize.root - розв'язування систем рівнянь

Для розв'язування систем нелінійних рівнянь чисельними методами застосовують функцію `scipy.optimize.root`. Її необов'язковий параметр `method` визначає метод розв'язування системи (`hybr`, `lm`, `df-sane`, `broyden1`, `broyden2`, `anderson`, `linearmixing`, `diagbroyden`, `excitingmixing`, `krylov`). За замовчуванням використовується `hybr`. В прикладі розв'язується система:

$$2x_0^2 + 2x_1 = 0;$$

$$x_0 - 2 = 0.$$

```
from scipy.optimize import root
def f(x, a, b, c): # векторна функція
    return [a*x[0]**2 + b*x[1],
            x[0]-c] # список лівих частин рівнянь
sol = root(f, [0, 0], args=(2,2,2)) # розв'язати
систему рівнянь з початковими значеннями коренів
x0=[0, 0]
print sol.x # корені
```

```
[ 2. -4.]
```

scipy.optimize.curve_fit - регресійний аналіз

Регресійний аналіз (http://en.wikipedia.org/wiki/Regression_analysis) - це статистичний метод дослідження впливу однієї або декількох незалежних змінних *x* на залежну змінну *y*. Для пошуку функціональної залежності *f(x)*, яка найкраще описує емпіричну залежність *y* від *x*, застосовують метод найменших квадратів (МНК). МНК оснований на мінімізації суми квадратів відхилень значень функції *f(x)* від значень *y*.

```
import numpy as np
from scipy.optimize import curve_fit, leastsq
```

```
import matplotlib.pyplot as plt # для побудови
графіків
x=np.array([0, 1, 2, 3]) # емпіричні значення x
y=np.array([1, 1.5, 3, 4]) # емпіричні значення y

# за допомогою scipy.optimize.curve_fit:
def f(x, a, b): # модель - лінійна залежність
y=a*x+b"
    return a*x+b # тут можна змінити модель на іншу
функцію
popt, pcov = curve_fit(f, x, y) # апроксимувати дані
залежністю f за допомогою нелінійного МНК
print popt # знайдені значення параметрів `a` i `b`
print pcov # коваріаційна матриця для `a` i `b`
print np.sqrt(np.diag(pcov)) # стандартні відхилення
для `a` i `b`
residuals = y - f(x,*popt) # відхилення
print "RMSE",(np.sum(residuals**2)/(residuals.size-
2))**0.5 # стандартна помилка (root-mean-square error
(RMSE))
ymean = np.mean(y) # середнє `y`
ss_res = np.dot(residuals, residuals)
ss_tot = np.dot((y-ymean), (y-ymean))
print "R^2:", 1-ss_res/ss_tot # коефіцієнт
детермінації (R-квадрат)
print "R^2:", np.corrcoef(y, f(x,*popt))[0,1]**2 #
або так

xa=np.linspace(0,3,100) # 100 значень на проміжку
0..3
ya=f(xa, *popt) # масив значень f з параметрами
a=popt[0], b=popt[1]
plt.plot(x, y, 'ko--') # нарисувати емпіричну
залежність
plt.plot(xa, ya, 'k-') # нарисувати апроксимовану
```

```
залежність
plt.xlabel('x'); plt.ylabel('y'); plt.show()
print "Рисунок - Лінійна регресія"

# або за допомогою scipy.optimize.leastsq:
def residuals_(p,x,y): # p - кортеж параметрів
    return y - f(x, p[0], p[1]) # повертає відхилення
ans=leastsq(func=residuals_,x0=(1,1),args=(x,y),full_
output=True) # мінімізує суму квадратів
print ans[0] # знайдені значення параметрів `a` і `b`

# або за допомогою scipy.stats.linregress (тільки
лінійна регресія):
from scipy.stats import linregress
slope, intercept, r_value, p_value,
std_err=linregress(x, y)
print slope, intercept # знайдені значення параметрів
лінійної залежності
print "R^2:", r_value**2 # коефіцієнт детермінації
(R-квадрат)
```

```
[ 1.05  0.8 ]
[[ 0.0175  -0.02625]
 [-0.02625  0.06125]]
[ 0.13228757  0.24748738]
RMSE 0.295803989155
R^2: 0.969230769231
R^2: 0.969230769231
```

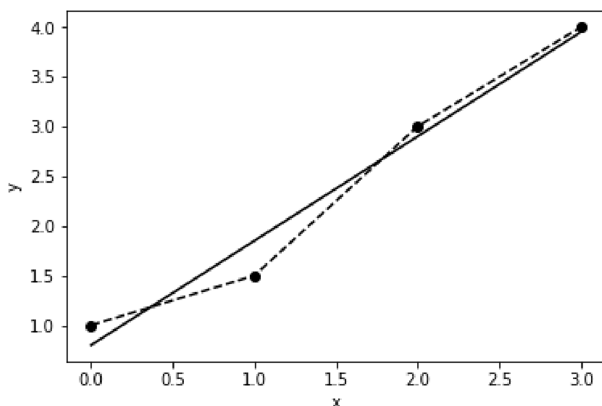


Рисунок 27 - Лінійна регресія

```
[ 1.05  0.8 ]
1.05 0.8
R^2: 0.969230769231
```

scipy.optimize.curve_fit - множинна регресія

Функція `scipy.optimize.curve_fit` може бути використана для апроксимації даних функцією багатьох змінних.

Таблиця 1 - Експериментальні дані - залежність *y* від *x0* і *x1*

x1 \ x0	0	1	2
0	0	1	2
1	1	2	3
2	2	3	7

```
import numpy as np
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

def f(x, a, b, c): # модель - функція двох змінних
```

```
x[0] і x[1]
    return a + b*x[0] + c*x[1]

# експериментальні дані:
x =
np.array([[0,1,2,0,1,2,0,1,2,],[0,0,0,1,1,1,2,2,2]])
# x0,x1
y = np.array([0,1,2,1,2,3,2,3,7]) # y
#y = np.array([0,1,2,1,2,3,2,3,4]) # спробуйте також
popt, pcov = curve_fit(f, x, y) # апроксимувати
нелінійним МНК
print popt # знайдені значення параметрів a, b, c

# рисуємо тривимірні графіки
from mpl_toolkits.mplot3d import axes3d # для
рисування 3D графіків
import matplotlib.pyplot as plt
fig = plt.figure() # створити фігуру
ax = fig.add_subplot(111, projection='3d') # додати
3D графік
# змінити форму масивів:
#
x0,x1=np.meshgrid(np.array([0,1,2]),np.array([0,1,2])
)
# або так:
x0=x[0].reshape((3,3)) # [[0 1 2],[0 1 2],[0 1 2]]
x1=x[1].reshape((3,3)) # [[0 0 0],[1 1 1],[2 2 2]]
y=y.reshape((3,3)) # [[0 1 2],[1 2 3],[2 3 7]]
print "R2=", r2_score(y, f((x0,x1),*popt)) #
коефіцієнт детермінації (R-квадрат)
ax.scatter(x0, x1, y) # показати емпіричні точки
xx0, xx1 = np.meshgrid(np.linspace(0, 2, 10),
np.linspace(0, 2, 10))
yy=f((xx0,xx1),*popt) # апроксимовані значення
ax.plot_wireframe(xx0, xx1, yy, rstride=1, cstride=1)
```

```
# показати поверхню
#ax.plot_surface(xx0, xx1, yy, rstride=1, cstride=1)
# або
ax.set_xlabel('x0');ax.set_ylabel('x1');ax.set_zlabel
('y');plt.show()
```

```
[-0.66666667  1.5         1.5       ]
R2= 0.678571428571
```

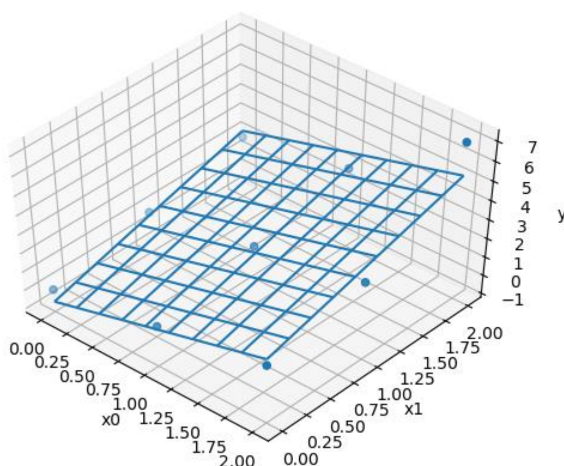


Рисунок 28 - Множинна регресія

scipy.optimize.fminbound - оптимізація функції однієї змінної з границями

В математиці оптимізацією називають задачу знаходження екстремуму (мінімуму або максимуму) функції *f(x)* в деякій області значень *x*. Оптимізація буває

- локальна - для пошуку локального екстремуму;
- глобальна - для пошуку глобального екстремуму.

Методи оптимізації поділяються на

- детерміновані;
- стохастичні;
- комбіновані.

За порядком похідної, що обчислюється, поділяються на

- прямі (обчислюються тільки значення функції);
- першого порядку (градієнтні);
- другого порядку.

Інтерфейсом для оптимізації функції однієї змінної різними методами є `minimize_scalar`. Розглянемо локальну оптимізацію скалярної функції (однієї змінної) $f(x) = sin(x) + cos(x^2)$ методом Брента.

```
import numpy as np
from scipy.optimize import minimize_scalar, fminbound
def f(x): # функція однієї змінної (цільова функція)
    return np.sin(x)+np.cos(x**2)
res = minimize_scalar(f, bounds=(-3, 3),
method='bounded') # знайти мінімум f(x) в заданих
границях. Критерій пошуку - мінімум, допустима
множина x від -3 до 3
print "argmin=",res.x

argmin = fminbound(f, -3, 3) # або так
print "argmin=", argmin
argmax = fminbound(lambda x: -f(x), -3, 3) # знайти
максимум f(x) в заданих границях
print "argmax=", argmax

import matplotlib.pyplot as plt
x=np.linspace(-3,3,100)
plt.plot(x,f(x),'k') # графік
plt.scatter([argmin,argmax], [f(argmin),f(argmax)],
```

```
linewidths=[3,3]) # локальні екстремуми
plt.xlabel('x');plt.ylabel('y');plt.grid();plt.show()
```

```
argmin= -1.75748900285
argmin= -1.75748900285
argmax= 0.730978404712
```

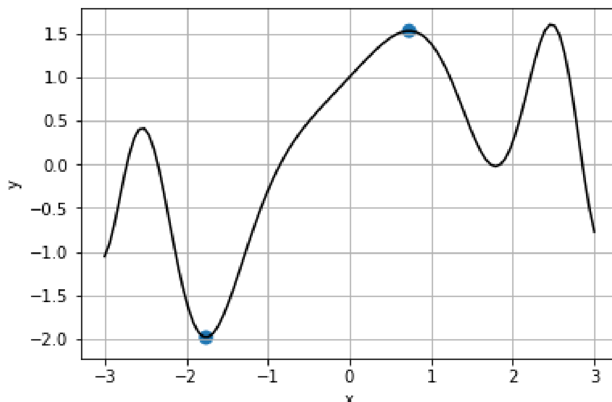


Рисунок 29 - Графік функції і знайдені локальні екстремуми

scipy.optimize.fminbound - локальна оптимізація невідомої функції

В цьому прикладі *f(x)* не розраховує значення наперед відомої функції і для кожного значення *x* користувач повинен ввести відповідне значення *y*, отримане, наприклад, експериментом. Для прикладу $y = x^2$, тоді на запит *x=2* користувач повинен ввести *y=4*. Користувач вводить *y* поки різниця нового і попереднього *x* є великою.

```
import numpy as np
from scipy.optimize import fminbound
i=0 # лічильник ітерацій
def f(x):
    global i # глобальна змінна
```

```
    i+=1 # збільшити лічильник
    y=input("Iteration %d x=%f y="%(i,x)) # вивести
`x` і ввести відповідне `y`
    return y
argmin = fminbound(f, -1, 1) # знайти мінімум
print argmin
```

```
Iteration 1 x=-0.236068 y=0.055728
Iteration 2 x=0.236068 y=0.055728
Iteration 3 x=0.527864 y=0.278640
Iteration 4 x=-0.000000 y=0.000000
Iteration 5 x=0.000003 y=0.000000
Iteration 6 x=0.000007 y=0.000000
```

scipy.optimize.fmin_l_bfgs_b - оптимізація з границями методом L-BFGS-B

Локальна оптимізація векторної функції (двох змінних) популярним квазі-ньютонівським методом L-BFGS-B, який призначений для нелінійних задач з великою кількістю невідомих (http://en.wikipedia.org/wiki/Limited-memory_BFGS). Інтерфейсом для оптимізації функції однієї або багатьох змінних різними методами є `minimize`. В прикладі шукається мінімум функції $y = x_0^2 + x_1^2$ в межах значень [-5, 5] змінних x_0 і x_1.

```
import numpy as np
from scipy.optimize import minimize, fmin_l_bfgs_b
def f(x): # функція двох змінних
    return x[0]**2+x[1]**2
res=minimize(f, x0=[1.0, 1.0], method="L-BFGS-B",
bounds=[(-5,5),(-5,5)])
print res.x
argmin = fmin_l_bfgs_b(f, x0=[1.0, 1.0], bounds=[(-
5,5),(-5,5)], approx_grad=True) # або так
print argmin[0]
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
ax=Axes3D(plt.figure()) # система координат
X, Y = np.meshgrid(np.linspace(-5,5), np.linspace(-
5,5)); Z=f([X,Y])
ax.plot_wireframe(X, Y, Z) # каркасна поверхня
ax.scatter(res.x[0], res.x[1], res.fun, c='k') #
мінімум
ax.set_xlabel('X0'),ax.set_ylabel('X1'),ax.set_zlabel
('Y');plt.show()
```

```
[ -5.01107818e-09  -5.01107818e-09]
[ -5.01107818e-09  -5.01107818e-09]
```

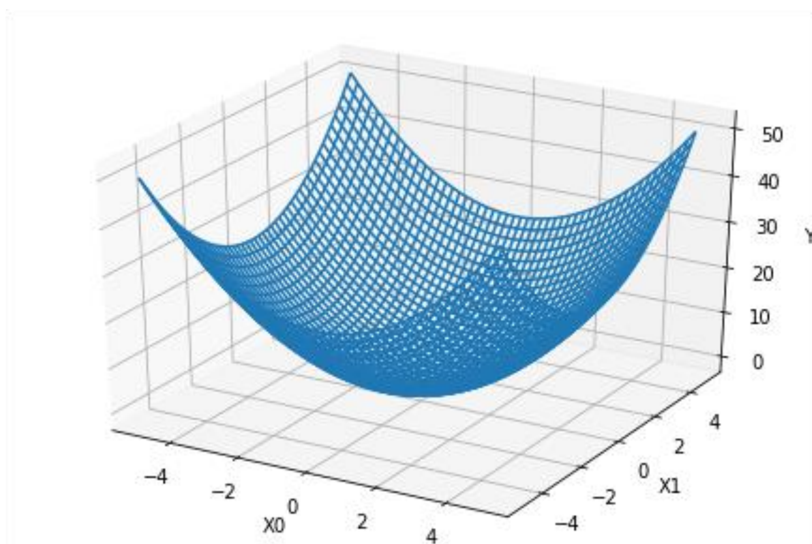


Рисунок 30 - Графік функції

scipy.optimize.differential_evolution - диференціальна еволюція

Диференціальна еволюція - це метод глобальної оптимізації функції однієї або багатьох змінних, який відноситься до стохастичних методів оптимізації з границями. Не використовує градієнтні методи, але потребує більшої кількості ітерацій. Моделює такі процеси біологічної еволюції як розмноження, мутація, рекомбінація і відбір. Доступні різні еволюційні стратегії (http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html). В прикладі шукається мінімум функції $x\sin x^2$ в межах [1, 4.5].

```
import numpy as np
from scipy.optimize import differential_evolution
f=lambda x: x*np.sin(x**2) # функція однієї змінної
res = differential_evolution(f, bounds=[(1, 4.5)]) #
знайти мінімум
print res

import matplotlib.pyplot as plt
x=np.linspace(1,5,200)
plt.plot(x,f(x),'k') # графік
plt.scatter(res['x'], res['fun'], linewidths=[3,3]) #
мінімум
plt.xlabel('x');plt.ylabel('y');plt.grid();plt.show()
```

```
     fun: array([-4.15851032])
     jac: array([  4.79616347e-06])
 message: 'Optimization terminated successfully.'
    nfev: 158
     nit: 9
 success: True
       x: array([ 4.16024526])
```

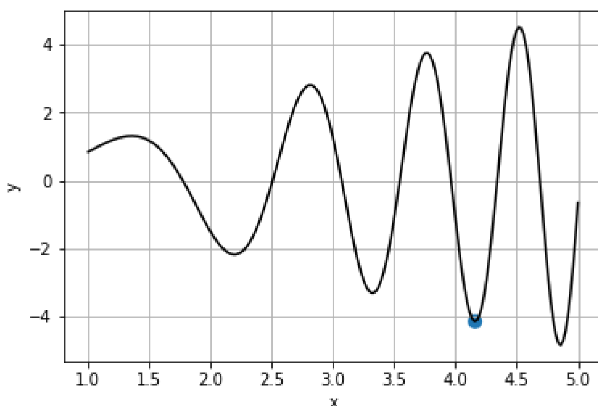


Рисунок 31 - Графік функції і знайдений мінімум в межах [1, 4.5]

scipy.optimize.basinhopping - комбінований метод глобальної оптимізації

Функція `scipy.optimize.basinhopping` реалізує комбінований метод глобальної багатомірної оптимізації. В кожній ітерації є етапи:

- Випадкове збурення координат.
- Локальна мінімізація.
- Прийняття або відхилення нових координати, основане на мінімізованому значенні функції.

Метод не гарантує знаходження мінімуму, оскільки алгоритм стохастичний. Алгоритм має багато параметрів, серед яких:

- `x0` - початкові значення мінімумів;
- `niter` - кількість ітерацій алгоритму;
- `T` - параметр для прийняття чи відхилення критерію;
- `stepsize` - початковий розмір кроку для випадкових зміщень;

- `minimizer_kwargs` - аргументи, які передаються локальному мінімізатору. Для пришвидшення можна також передавати мінімізатору похідні функції (Якобіан, Гессіан);
- `take_step` - замінює функцію кроку за замовчуванням;
- `accept_test` - визначає тест, який використовується для прийняття або відхилення кроку;
- `callback` - функція, яка викликається, коли знайдено локальний мінімум.

```
import numpy as np
from scipy.optimize import basinhopping

def f(x): # функція двох змінних
    y = np.cos(14.5*x[0]-
0.3)+(x[1]+0.2)*x[1]+(x[0]+0.2)*x[0]
    #y=1 + 2*x[0] + 2*x[1] # (тільки для тестування
accept_test. Див. нижче)
    return y # для пришвидшення пошуку мінімуму
функція разом зі значенням може також повертати
градієнт (див. документацію)

ret = basinhopping(func=f, x0=[0.0, 0.0]) # знайти
мінімум
print ret['x'], ret['fun']

class MyBounds(object): # реалізує границі проблеми
    def __init__(self, xmax=[1.1, 1.1], xmin=[-1.1, -
1.1] ):
        self.xmax = np.array(xmax) # максимум для x
        self.xmin = np.array(xmin) # мінімум для x
    def __call__(self, **kwargs):
        x = kwargs["x_new"]
        tmax = bool(np.all(x <= self.xmax)) # True,
якщо x менше рівне максимуму
        tmin = bool(np.all(x >= self.xmin)) # True,
```

```
якщо x більше рівне мінімуму
        return tmax and tmin # якщо False (за
границями), то мінімум не приймається

def print_fun(x, f, accepted): # викликається, коли
знайдено лок. мінімум
    print "Loc. min.", x, f, accepted # координати,
значення функції, чи мінімум приймається

ret = basinhopping(func=f, x0=[0.0, 0.0], niter=5,
stepsize=0.1, minimizer_kwargs = {"method": "L-BFGS-
B"}, accept_test=MyBounds(), callback=print_fun) #
знайти мінімум
print ret['x'], ret['fun']

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
ax=Axes3D(plt.figure()) # система координат
X, Y = np.meshgrid(np.linspace(-1,1), np.linspace(-
1,1)); Z=f([X,Y])
ax.plot_wireframe(X, Y, Z) # каркасна поверхня
ax.scatter(ret['x'][0], ret['x'][1], ret['fun'],
c='k') # мінімум
ax.set_xlabel('X0'),ax.set_ylabel('X1'),ax.set_zlabel
('Y');plt.show()
```

```
[-0.19506755 -0.10000001] -1.01087618444
Loc. min. [-1.05352142 -0.09999747] -0.102110348616 T
rue
Loc. min. [-0.19506756 -0.1       ] -1.01087618444 Tr
ue
Loc. min. [-0.19506756 -0.1       ] -1.01087618444 Tr
ue
Loc. min. [ 0.23417127 -0.10000023] -0.907266719691 T
rue
```

```
Loc. min. [ 0.23417128 -0.10000001] -0.907266719691 T
rue
[-0.19506756 -0.1       ] -1.01087618444
```

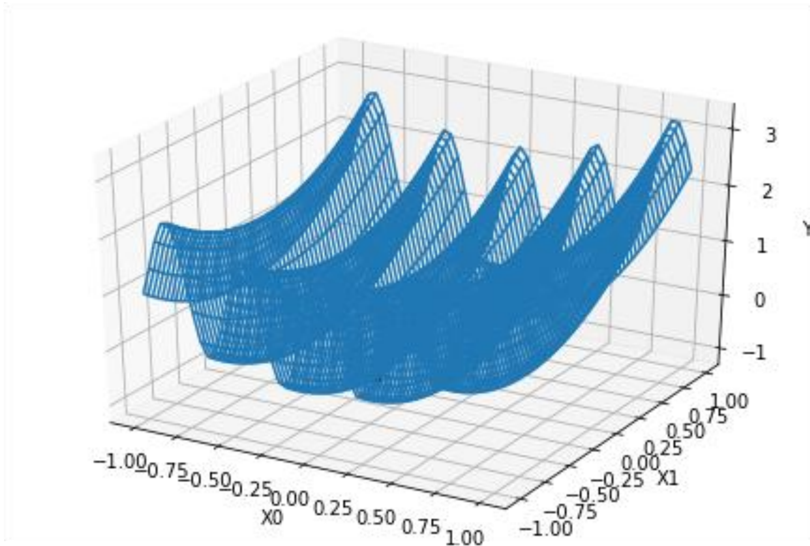


Рисунок 32 - Графік функції

scipy.stats - випадкові величини

Модуль `scipy.stats` містить велику кількість дискретних і неперервних розподілів імовірності і бібліотеку статистичних функцій. Випадкова величина (ВВ) - це величина, значення якої є результатом випадкового явища. Розподіл ймовірностей - це закон, який описує область значень випадкової величини і імовірності їх появи. В прикладі показані функції для роботи з ВВ.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

```
dist = stats.norm(loc=15, scale=1) # ВВ з нормальним розподілом
X = np.linspace(10,20,100) # область значень ВВ
Xs=dist.rvs(size=1000) # випадкова вибірка значень ВВ розміром 1000
print Xs.mean(), Xs.std(), Xs.var() # середнє, середньоквадратичне відхилення, дисперсія
plt.hist(Xs, bins=20, normed=True, color='y') # гістограма вибірки
plt.plot(X, dist.pdf(X),'k') # функція густини імовірності розподілу ВВ
plt.plot(X, dist.cdf(X),'k--') # функція розподілу ВВ (імовірність того, що ВВ буде мати значення менше або рівне x)
print dist.cdf(15)-dist.cdf(0) # імовірність попадання в інтервал значень (0,15)
plt.xlabel('x');plt.ylabel('y, Y');plt.grid();plt.show()
print "Рисунок - Гістограма, функція густини імовірностей y(x) (-) та функція розподілу Y(x) (--)"

plt.figure()
Y=np.linspace(0,1,100) # область значень функції розподілу
plt.plot(Y, dist.ppf(Y),'k') # квантільна функція (інверсна до cdf)
print dist.ppf(0.5) # значення ВВ, якому відповідає cdf=0.5
cdf1=stats.norm().cdf(3) # 0.998650101968
cdf2=stats.norm().cdf(-3) # 0.00134989803163
print cdf1-cdf2 # імовірність попадання в інтервал (-3*std,+3*std)
# квантілі стандартного нормального розподілу:
print stats.norm().ppf(cdf1) # з рівнем cdf1
print stats.norm().ppf(0.5) # з рівнем 0.5
```

```
plt.xlabel('Y');plt.ylabel('x');plt.grid();plt.show()
print "Рисунок - Квантільна функція x(Y)"
```

```
14.9819184126 0.984759375625 0.969751027882
0.5
```

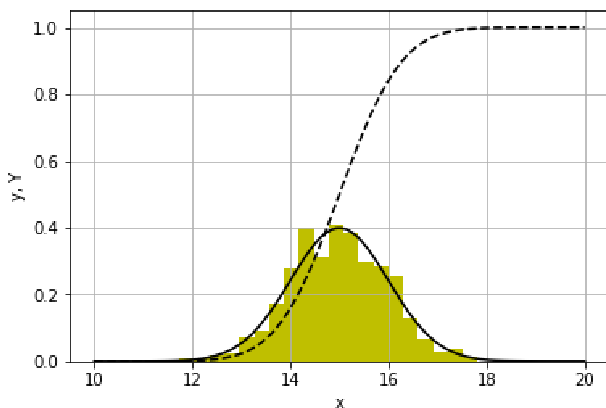


Рисунок 33 - Гістограма, функція густини ймовірностей *y(x)* (-) та функція розподілу *Y(x)* (--)

```
15.0
0.997300203937
3.0
0.0
```

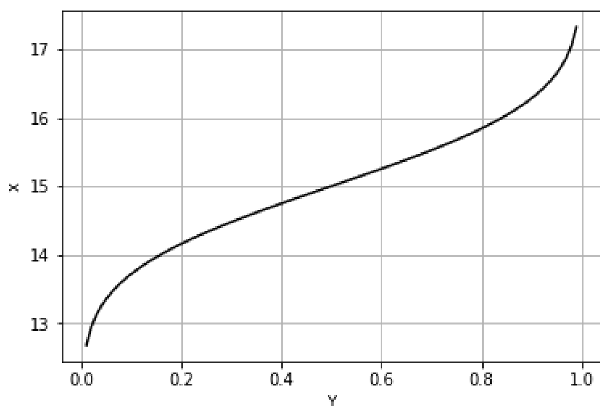


Рисунок 34 - Квантільна функція *x(Y)*

scipy.stats - підгонка кривих і перевірка статистичних гіпотез

Дано вибірку з *N* значеннями випадкової величини. Потрібно вияснити чи підлягає ця випадкова величина нормальному розподілу. Це можна зробити за допомогою:

- візуального порівняння емпіричної гістограми і кривої нормального розподілу;
- функції `normaltest`;
- функції `kstest`;
- функції `chisquare`.

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

dist = stats.norm(loc=15, scale=0.5) # нормальний розподіл
#dist=stats.uniform(loc=14,scale=2) # рівномірний розподіл (для порівняння)
N=1000 # розмір вибірки
```

```
X=dist.rvs(size=N) # випадкова вибірка з цього
розподілу
mean,std = stats.norm.fit(X) # підігнати криву норм.
розп. і отримати її параметри
k2,pvalue = stats.normaltest(X) # тест на нормальний
розподіл (k2 - сума квадратів коефіцієнтів асиметрії
і ексцесу)
print k2,pvalue # наприклад, якщо pvalue < 0.05, то
це не нормальний розподіл
d,pvalue = stats.kstest(X, dist.cdf) # тест
Колмогорова-Смірнова
#або stats.kstest(X, 'norm', args=(15, 0.5))
print d,pvalue # якщо pvalue < 0.05, то ці розподіли
не ідентичні

# побудова емпіричної гістограми і теоретичної
кривої:
n,x=np.histogram(X, bins=10) # кількість значень в
кожному інтервалі, інтервали
xmin=x[:-1] # масив мінімумів інтервалів
dx=x[1]-x[0] # ширина інтервалу
y=n/(N*dx) # емпірична приведена частота
# площа одного прямокутника гістограми дорівнює
відносній частоті: p=n/N=dx*n/(N*dx)
xmid=xmin+dx/2 # масив центрів інтервалів
plt.bar(xmid, y, width=dx, color='y') # нарисувати
емпіричну гістограму приведених частот
dist2 = stats.norm(loc=mean, scale=std) # нормальний
розподіл з параметрами після підгонки
plt.plot(xmid, dist2.pdf(xmid),'ko') # нарисувати
точки теоретичної кривої
X1=np.linspace(13,17,100) # аргументи для побудови
теоретичної кривої
plt.plot(X1, dist2.pdf(X1),'k-') # нарисувати
теоретичну криву густини імовірності
```

```
plt.xlabel('x');plt.ylabel('y');plt.grid();plt.show()
print "Рисунок - Гістограма і функція густини 
розподілу"
# перевірка:
print np.sum(dx*y) # сума площ прямокутників: 1.0 
,або sum(n/N)
print np.trapz(dist2.pdf(X1), X1) # інтеграл pdf: 1.0

# порівнюємо емпіричні і теоретичні абсолютні частоти
n1=np.diff(dist2.cdf(x))*N # теоретичні абсолютні 
частоти - це площі криволінійних трапецій на кожному 
інтервалі. Без множення на N буде близько 1.0
plt.figure()
plt.plot(n,'ks--',n1,'ko-') # емпіричні і теоретичні 
абсолютні частоти
plt.xlabel(u'інтервал');plt.ylabel('n');plt.grid();pl
t.show()
print "Рисунок - Емпіричні (--) і теоретичні (-) 
абсолютні частоти"
chisq, p_value = stats.chisquare(n, n1) # хи-квадрат 
тест гіпотези подібності частот n і n1
print chisq, p_value # гіпотеза приймається, якщо 
p_value більше заданого (0.05)
```

```
6.18744549283 0.0453328770782
0.0245207935549 0.584607787303
```

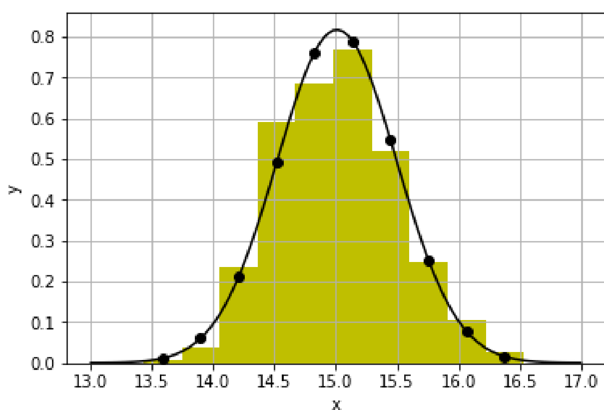


Рисунок 35 - Гістограма і функція густини розподілу

```
1.0
0.999957666033
```

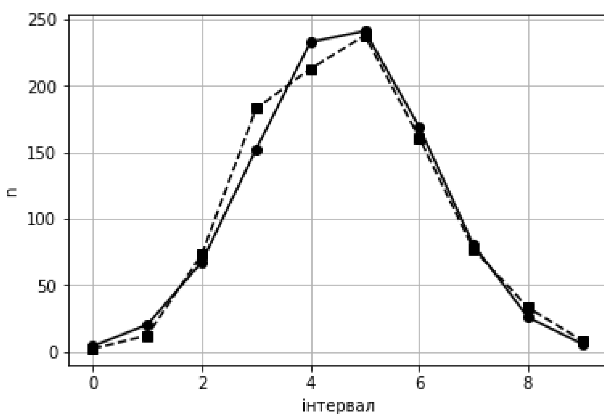


Рисунок 36 - Емпіричні (--) і теоретичні (-) абсолютні частоти

```
16.4635646646 0.0578098854704
```

scipy.stats.kde - ядрова оцінка густини розподілу

Ядрова оцінка густини розподілу - це непараметричний метод оцінки функції густини випадкової величини за вибіркою (http://en.wikipedia.org/wiki/Kernel_density_estimation). Задається формулою

$$f(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right),$$

де x_i - значення незалежних і однаково-розподілених випадкових величин; h - параметр згладжування; K - статистичне ядро - симетрична, але не обов'язково додатна функція з інтегралом рівним одиниці. В прикладі використовується Гаусове ядро:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-0.5u^2}.$$

```
import numpy as np
from scipy.stats import kde
import matplotlib.pyplot as plt
x1 = np.random.normal(0, 3, 50) # вибірка з
нормального розподілу розміром 50, m=0, std=3
x2 = np.random.normal(4, 1, 50) # вибірка з
нормального розподілу розміром 50, m=4, std=1
x = np.concatenate([x1,x2]) # об'єднати дані
density = kde.gaussian_kde(x, bw_method=None) #
функція щільності. Можна також визначити свій метод
згладжування bw_method. Може бути багатовимірна.
xgrid = np.linspace(x.min(), x.max(), 100)
plt.hist(x, bins=8, normed=True, color='y') #
гістограма
plt.plot(xgrid, density(xgrid), 'k-') # функція
щільності
plt.xlabel('x');plt.ylabel('y');plt.grid();plt.show()
```

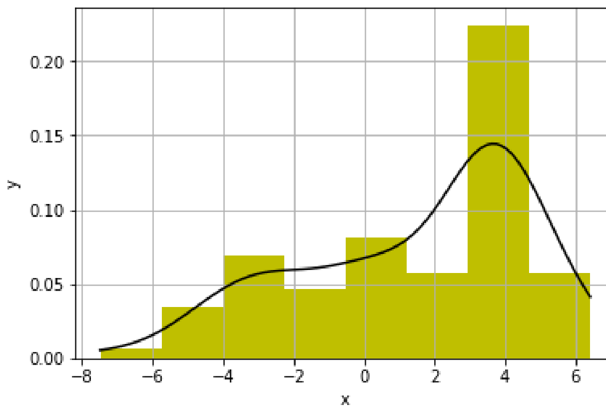


Рисунок 37 - Ядрова оцінка густини розподілу

scipy.fftpack дискретне перетворення Фур'є

Перетворення Фур'є — інтегральне перетворення однієї комплекснозначної функції дійсної змінної на іншу. Розкладає функцію на осциляторні функції, тобто подає сигнал у вигляді суми гармонічних коливань. Використовується для розрахунку спектра частот сигналів, які змінні у часі. Для дискретних функцій застосовують дискретне перетворення Фур'є. Модуль `scipy.fftpack` використовує швидкі алгоритми прямого і оберненого дискретного перетворення Фур'є (FFT).

```
import numpy as np
from scipy.fftpack import fft
import matplotlib.pyplot as plt
N = 600 # кількість точок
T = 1.0 / 800.0 # проміжки часу
A1=1.0; A2=0.5 # амплітуди
f1=10.0; f2=100.0 # лінійні частоти
omega1=f1*2.0*np.pi; omega2=f2*2.0*np.pi # циклічні частоти
x = np.linspace(0.0, N*T, N) # час
```

```
y = A1*np.sin(omega1*x) + A2*np.sin(omega2*x) # сигнал (лінійна комбінація синусоїдальних сигналів)
plt.plot(x,y,'k') # нарисувати сигнал
plt.xlabel('t');plt.ylabel('y');plt.grid();plt.show()
print "Рисунок - Сигнал"

yf = fft(y) # розрахувати спектр частот для сигналів змінних у часі за допомогою дискретного перетворення Фур'є. Декомпозиція сигналу на частоти і амплітуди.
xf = np.linspace(0.0, 1.0/(2.0*T), N/2) # частоти
yf_=2.0/N * np.abs(yf[0:N/2]) # амплітуди
plt.figure()
plt.plot(xf, yf_,'k') # нарисувати спектр
plt.xlabel('f');plt.ylabel('A');plt.grid();plt.show()
print "Рисунок - Амплітудно-частотна характеристика сигналу"
```

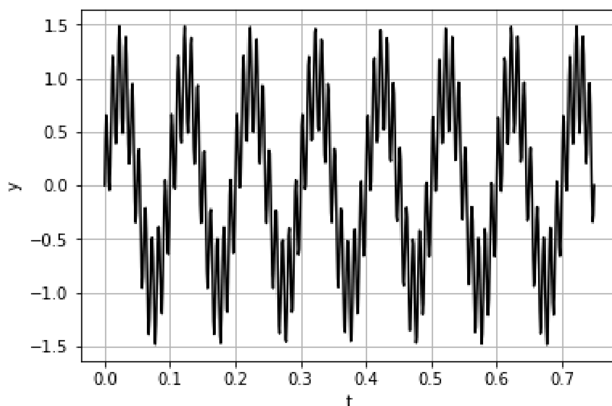


Рисунок 38 - Сигнал

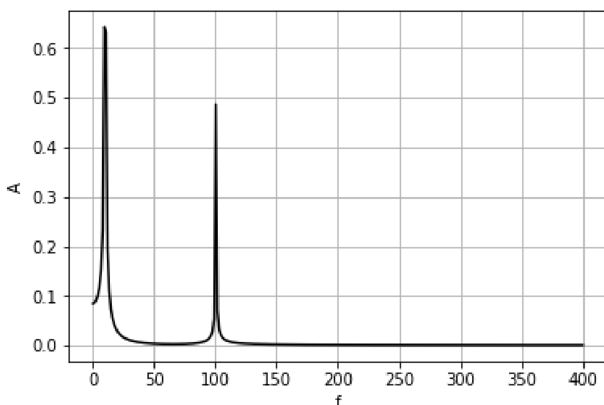


Рисунок 39 - Амплітудно-частотна характеристика сигналу

scipy.fftpack - обернене дискретне перетворення Фур'є

Обернене дискретне перетворення Фур'є повертає сигнал за спектром його частот. В прикладі показано застосування прямого і оберненого дискретного перетворення Фур'є для частотного фільтрування сигналу.

```
import numpy as np
from scipy.fftpack import rfft, irfft, fftfreq
import matplotlib.pyplot as plt
time   = np.linspace(0,2,2000) # час
signal = np.cos(5*np.pi*time) +
2*np.cos(7*np.pi*time) # сигнал
W = fftfreq(signal.size, d=time[1]-time[0]) # частоти
f_signal = rfft(signal) # спектр (дискретне
перетворення Фур'є для дійсних)
cut_f_signal = f_signal.copy() # копія сигналу
cut_f_signal[(W<6)] = 0 # фільтруємо сигнал
(відкидаємо частоти<6)
cut_signal = irfft(cut_f_signal) # відфільтрований
сигнал (обернене дискретне перетворення Фур'є для
```

```
дійсних)

plt.subplot(121); plt.plot(time,signal);
plt.xlabel('t'); plt.ylabel('y')
plt.subplot(122); plt.plot(time,cut_signal);
plt.xlabel('t')
plt.show(); print "Рисунок - Початковий і
відфільтрований сигнал"

plt.figure()
plt.subplot(121); plt.plot(W,f_signal);
plt.xlim(0,10); plt.xlabel('f'); plt.ylabel('A')
plt.subplot(122); plt.plot(W,cut_f_signal);
plt.xlim(0,10); plt.xlabel('f')
plt.show(); print "Рисунок - Спектр початкового і
відфільтрованого сигналу"
```

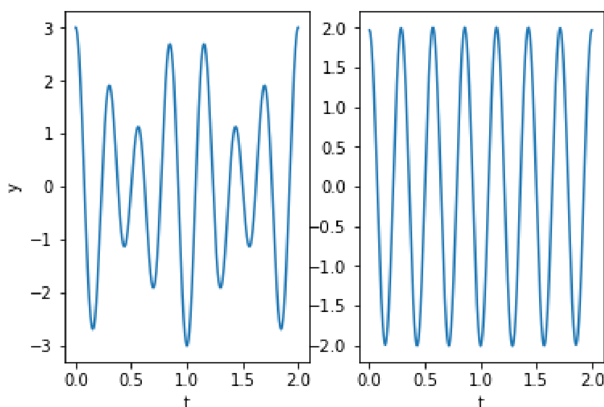


Рисунок 40 - Початковий і відфільтрований сигнал

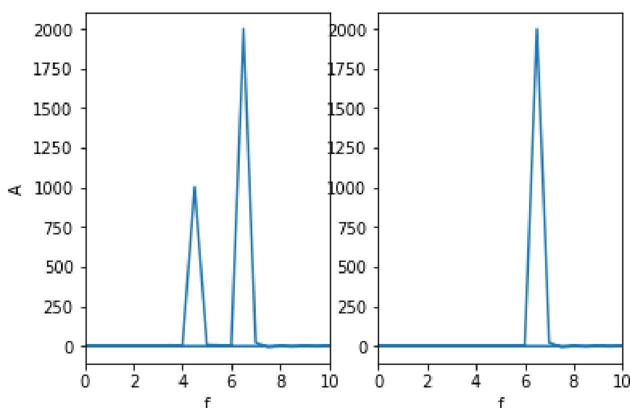


Рисунок 41 - Спектр початкового і відфільтрованого сигналу

scipy.cluster - кластеризація

Кластеризація (кластерний аналіз) - задача розбиття множини об’єктів на групи (кластери) подібних об’єктів. В прикладі показана кластеризація методом k-середніх, який оснований на мінімізації сумарного квадратичного відхилення точок кластерів від центрів цих кластерів. Спробуйте також потужній алгоритм кластеризації даних з наявністю шуму `sklearn.cluster.DBSCAN`.

```
import numpy
from scipy.cluster import vq
import matplotlib.pyplot as plt

# масиви випадкових точок з координатами x,y
c1 = numpy.random.randn(100, 2) + 5
c2 = numpy.random.randn(30, 2) - 5
c3 = numpy.random.randn(50, 2)
data = numpy.vstack([c1, c2, c3]) # об'єднати дані

whiten=vq.whiten(data) # нормалізувати дані
centroids,labels=vq.kmeans2(whiten,3) # центроїди і
```

```
мітки 3-х кластерів
#plt.scatter(data[:,0],data[:,1],c=labels);
plt.show()

# або
centroids, distortion = vq.kmeans(data, 3) #
centroids - центроїди 3-х кластерів методом k-
середніх, distortion - сума квадратів відстаней між
точками і відповідною центроїдою
identified, distance = vq.vq(data, centroids) # масив
для ідентифікації та масив з відстанями до центроїди
# координати точок та відстані до центроїди в кожному
кластері
vqc1,d1 = data[identified == 0], distance[identified
== 0]
vqc2,d2 = data[identified == 1], distance[identified
== 1]
vqc3,d3 = data[identified == 2], distance[identified
== 2]
plt.figure()
plt.scatter(vqc1[:,0],vqc1[:,1],c='r',s=d1*20) #
точки кластера 1
plt.scatter(vqc2[:,0],vqc2[:,1],c='g',s=d2*20) #
точки кластера 2
plt.scatter(vqc3[:,0],vqc3[:,1],c='b',s=d3*20) #
точки кластера 3
plt.scatter(centroids[:,0], centroids[:,1],
marker='o', s=10000, c='k', alpha=0.2) # центроїди
plt.xlabel("x");plt.ylabel("y");plt.grid();plt.show()
```

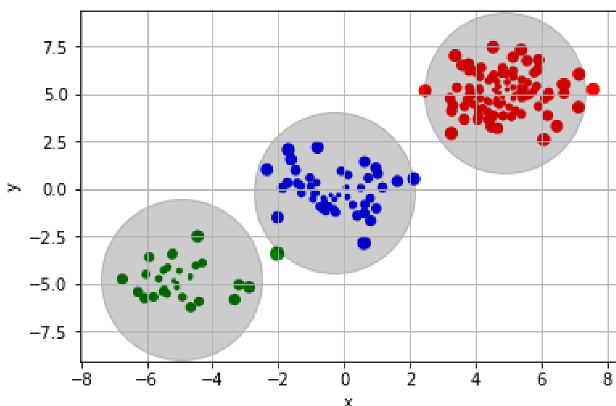


Рисунок 42 - Результати кластеризації

pandas - аналіз даних

pandas (http://pandas.pydata.org) - бібліотека, яка базується на NumPy і містить високопродуктивні та зручні у використанні структури даних та інструменти обробки і аналізу даних. За функціональністю pandas подібна на табличний процесор Excel. Основними структурами даних є `Series` (одновимірний масив ndarray з мітками осі) та `DataFrame` (таблиця з мітками осей (рядків і стовпців)). Приклад описує основні можливості pandas 0.20.3.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
_='\n'
x1 = [0, 2, 2, 3, 9]
x2 = [12, 12, None, 20, 31]
dataSet = zip(x1,x2) # підготувати дані
df = pd.DataFrame(data = dataSet, columns=['X1',
'X2']) # об'єкт DataFrame
sr = pd.Series([1,3,np.nan,7,9]) # об'єкт Series
print df,_ # вивести таблицю
```

```
#print df.head() # вивести початок таблиці
#df.to_csv('rodStats.csv',index=False,header=False) #
зберегти у файл csv
#df = pd.read_csv('rodStats.csv',names=['X1', 'X2'])
# прочитати з файлу csv

print df.dtypes,_ # типи даних колонок
print df.X1,_ # вміст колонки (Series)
#df['X1'] # або
print df['X1'].unique(),_ # унікальні значення
колонки
df[0:2] # перші 2 рядка (DataFrame)
print df.loc[:,'X1'],_ # індексування (Series містить
тільки X1)
#df[['X1']] # або (DataFrame містить тільки X1)
print df[df['X1'] == 0],_ # умовне індексування
(DataFrame)
# або:
df[(df['X1'] == 0) & (df['X2'] > 0)] # | - or, & -
and, ~ - not
df['X2'][df['X1'] == 0] # (Series)

print df['X1'].values,_ # конвертація в numpy.ndarray
df.dropna() # відкинути рядки з відсутніми даними
(None) (див. також fillna)
df.sort_index(axis=1) # сортувати за колонками (1)
або рядками (0)
df.sort_values(['X1'], ascending=False) # сортувати
за X1 (за спаданням)
df['X3'] = np.sqrt(df['X1']**2+df['X2']**2) # додати
нову колонку
print df['X3'].map(lambda x: x+1),_ # застосувати
функцію для кожного елемента Series (див. також apply
і applymap для DataFrame)
print df.groupby(df['X1']).mean(),_ # групувати за X1
```

```
і знайти середнє в групах
print df.pivot(index='X1', columns='X2',
values='X3'),_ # звідна таблиця, де X1 - рядки, X2 -
колонки, X3 - значення (див. також pd.pivot_table)
print df.stack(),_ # ієрархічне представлення даних
(див. також unstack, MultiIndex - ієрархічний індекс)
print pd.crosstab(df['X1'], df['X2']),_ # таблиця
частот факторів X1,X2

print df.describe(),_ # статистика для кожної колонки
(див. також mean, std, ...)
print df.cov(),_ # коваріація (математичне сподівання
добутків відхилень випадкових величин)
print df.corr() # кореляція (коеф. корел. Пірсона =
covXY/(Sx*Sy))

#df.plot(kind='bar') # візуалізація
#df.plot(x='X1', y='X2')
#plt.show()
```

```
   X1    X2
0   0  12.0
1   2  12.0
2   2   NaN
3   3  20.0
4   9  31.0

X1       int64
X2     float64
dtype: object

0    0
1    2
2    2
3    3
```

```
4    9
Name: X1, dtype: int64

[0 2 3 9]

0    0
1    2
2    2
3    3
4    9
Name: X1, dtype: int64

   X1    X2
0   0  12.0

[0 2 2 3 9]

0    13.000000
1    13.165525
2          NaN
3    21.223748
4    33.280025
Name: X3, dtype: float64

    X1    X2         X3
X1
0    0  12.0  12.000000
2    2  12.0  12.165525
3    3  20.0  20.223748
9    9  31.0  32.280025

X2  NaN         12.0       20.0       31.0
X1
0     NaN  12.000000        NaN        NaN
2     NaN  12.165525        NaN        NaN
```

```
3      NaN        NaN  20.223748        NaN
9      NaN        NaN        NaN  32.280025

0  X1     0.000000
   X2    12.000000
   X3    12.000000
1  X1     2.000000
   X2    12.000000
   X3    12.165525
2  X1     2.000000
3  X1     3.000000
   X2    20.000000
   X3    20.223748
4  X1     9.000000
   X2    31.000000
   X3    32.280025
dtype: float64

X2  12.0  20.0  31.0
X1
0      1     0     0
2      1     0     0
3      0     1     0
9      0     0     1

             X1         X2         X3
count  5.000000   4.000000   4.000000
mean   3.200000  18.750000  19.167325
std    3.420526   8.995369   9.547333
min    0.000000  12.000000  12.000000
25%    2.000000  12.000000  12.124144
50%    2.000000  16.000000  16.194637
75%    3.000000  22.750000  23.237818
max    9.000000  31.000000  32.280025
```

```
          X1         X2         X3
X1  11.700000  33.500000  35.726658
X2  33.500000  80.916667  85.864232
X3  35.726658  85.864232  91.151559

         X1        X2        X3
X1  1.000000  0.961568  0.966195
X2  0.961568  1.000000  0.999796
X3  0.966195  0.999796  1.000000
```

scikit-learn - машинне навчання

Бібліотека scikit-learn (http://scikit-learn.org) містить зручні для використання алгоритми машинного навчання з учителем (регресія, класифікація) і без учителя (кластеризація, зменшення розмірності), а також засоби для підготовки даних і вибору найкращої моделі даних [43]. В прикладі за допомогою scikit-learn 0.19.1 розв'язується задача бінарної класифікації. Необхідно навчити класифікатор `RandomForestClassifier` розпізнавати класи *y* нових даних *x1*, *x2*. Для цього модель навчається на даних для навчання. Тестові дані використовуються для перевірки правильності роботи моделі на нових даних. Як видно, ця модель правильно визначає класи 100% тестових даних.

Але така однократна процедура не є надійною. Для надійної перевірки правильності в прикладі застосовується 3-х блокова перехресна перевірка моделі. Вона виконується шляхом поділу усіх даних на 3 частини (*xy1*, *xy2*, *xy3*). Далі для кожної частини функція `cross_val_score` виконує навчання моделі і розрахунок правильності на незадіяних для навчання частинах. Тепер середнє значення правильності - 85 %.

Окремою проблемою є визначення оптимальних значень параметрів моделі `n_estimators` і `max_depth`. Якщо кількість їх варіантів не велика, то можна виконати цю програму для різних значень і подивитись, у якому випадку правильність найвища.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

x=np.array([[0,1,1,2,2,3,2,3,1,3,
6,5,6,7,7,8,7,7,8,5],
            [1,1,3,1,2,2,3,4,4,8,
5,7,6,7,6,7,5,8,8,1]]) # дві ознаки класів
y=np.array( [0,0,0,0,0,0,0,0,0,0,
1,1,1,1,1,1,1,1,1,1] ) # мітки класів (бінарна
класифікація)
x=x.T
plt.scatter(x[:,0], x[:,1], c=y, s=100) #
візуалізація класів
plt.xlabel('x0'); plt.ylabel('x1'); plt.show()
print "Рисунок - Два класи даних"
# розбити дані (дані для навчання і тестові дані для
перевірки)
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.5)

model=RandomForestClassifier(n_estimators=5,
max_depth=3) # модель - випадковий ліс (n_estimators
- кількість дерев, max_depth - глибина дерева)
model.fit(x_train, y_train) # виконати навчання
print y_test # фактичні тестові класи
print model.predict(x_test) # прогнозовані тестові
класи
print model.score(x_test, y_test) # правильність
класифікатора

# перехресна перевірка (удосконалення
train_test_split + score)
from sklearn.model_selection import cross_val_score
```

```
s=cross_val_score(model, x, y, cv=3)
print s, s.mean() # правильність класифікатора на кожній ітерації і її середнє значення
```

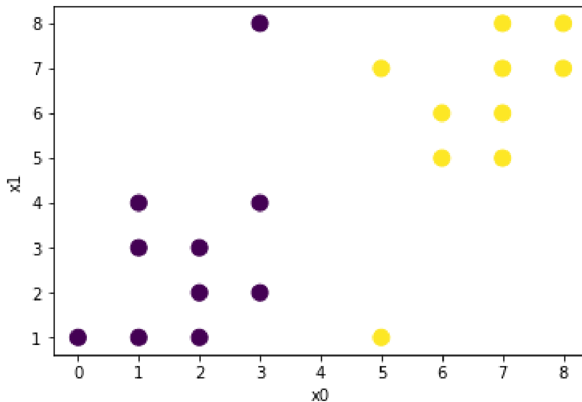


Рисунок 43 - Два класи даних

```
[1 0 0 0 1 0 0 0 1 1]
[1 0 0 0 1 0 0 0 1 1]
1.0
[ 0.875       1.          0.66666667] 0.847222222222
```

NetworkX - графи

NetworkX (http://networkx.github.io) - це пакет для створення, маніпуляції і вивчення структури, динаміки і функціонування комплексних графів. Граф - це абстрактний математичний об'єкт, який являє собою множину вершин і ребер, які з'єднують пари вершин. В прикладі показані основи створення і використання неорієнтованих графів за допомогою NetworkX 2.0.

```
import networkx as nx
import matplotlib.pyplot as plt
```

```
G=nx.Graph() # створити неорієнтований граф
G.add_node(1) # додати вузол
G.add_node('A') # додати вузол (вузлом може бути
будь-який об'єкт)
G.add_nodes_from([2,3,4]) # додати вузли
G.add_edge(1,2) # додати ребро
G.add_edges_from([(2,3),(3,4),(4,2),(2,'A')]) #
додати ребра

print G.number_of_nodes(), G.number_of_edges() #
кількість вузлів і ребер
print 'nodes', G.nodes # вузли
print 'edges', G.edges # ребра
print 'adj', G.adj # сусіди вершин
print 'degree', G.degree # степені вершин (кількість
ребер вершин)

print G.edges(['A',2,3]) # усі ребра вершин 'A',2,3
print G[2] # сусіди вершини 2, або G.adj[2]
print G.degree(['A',2,3]) # степені вершин 'A',2,3

G.node['A']['a']="val1" # змінити значення атрибута
'a' вузла 'A'
print G.nodes['A'] # словник атрибутів вузла

G[1][2]["color"]="blue" # змінити значення атрибута
'color' ребра 1,2
# або G.edges[1,2]["color"]="blue"
G[1][2]['weight'] = 4 # спеціальний атрибут зважених
графів
print G[1][2] # словник атрибутів ребра 1,2

# ітерація по кортежам (node, neighbors), де
neighbors - словник:
for node, neighbors in G.adj.items():
```

```
    print "Сусіди вузла", node
    for neighbor, edge_attr in neighbors.items(): # для кожного сусіда
        print '    ', neighbor, edge_attr # сусід, властивості ребра

#nx.draw(G, with_labels = True) # рисувати граф за допомогою matplotlib
nx.draw_circular(G, with_labels = True) # інші способи візуалізації графа
#nx.draw_spectral(G, with_labels = True)
plt.show(); print "Рисунок - Візуалізація графа"

#якщо Graphviz і PyGraphviz (nx_agraph) або pydot (nx_pydot) установлені, то можна рисувати, зберігати і читати граф у форматі dot. Підтримуються також інші формати.
#nx.draw(G, pos=nx.nx_agraph.graphviz_layout(G)) # рисувати
#nx.drawing.nx_agraph.write_dot(G, "myGraph.dot") # зберегти
#G=nx.drawing.nx_agraph.read_dot("myGraph.dot") # прочитати
```

```
5 5
nodes ['A', 1, 2, 3, 4]
edges [('A', 2), (1, 2), (2, 3), (2, 4), (3, 4)]
adj {'A': {2: {}}, 1: {2: {}}, 2: {'A': {}, 1: {}, 3: {}, 4: {}}, 3: {2: {}, 4: {}}, 4: {2: {}, 3: {}}}
degree [('A', 1), (1, 1), (2, 4), (3, 2), (4, 2)]
[('A', 2), (2, 1), (2, 3), (2, 4), (3, 4)]
{'A': {}, 1: {}, 3: {}, 4: {}}
[('A', 1), (2, 4), (3, 2)]
{'a': 'val1'}
{'color': 'blue', 'weight': 4}
```

```
Сусіди вузла A
      2 {}
Сусіди вузла 1
      2 {'color': 'blue', 'weight': 4}
Сусіди вузла 2
      A {}
      1 {'color': 'blue', 'weight': 4}
      3 {}
      4 {}
Сусіди вузла 3
      2 {}
      4 {}
Сусіди вузла 4
      2 {}
      3 {}
```

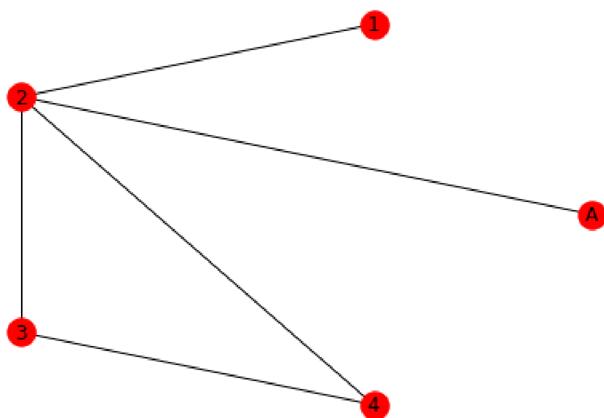


Рисунок 44 - Візуалізація графа

NetworkX - орієнтовані графи, алгоритми на графах

Ребра графа, які мають напрямок, називають дугами. Неорієнтований граф містить тільки ребра, а орієнтований граф містить тільки дуги. В NetworkX орієнтовані графи створюються за допомогою класу `DiGraph`. В прикладі показані операції з орієнтованими графами і розповсюджені алгоритми на графах (http://networkx.github.io/documentation/stable/reference/algorithms/index.html).

```
import networkx as nx
import matplotlib.pyplot as plt
G=nx.DiGraph() # створити орієнтований граф
G.add_weighted_edges_from([(1,2,0.5), (1,3,0.5),
(3,4,0.25)]) # додати ребра з вагами

print 'suc', list(G.successors(1)) # вузли-нащадки
print 'pre', list(G.predecessors(1)) # вузли-предки
print 'nei', list(G.neighbors(1)) # вузли-сусіди
print 'out', G.out_edges(1) # вихідні ребра
print 'in', G.in_edges(1) # вхідні ребра

print G.degree(1) # кількість ребер цього вузла. Але:
print G.degree(1, weight='weight') # сума ваг ребер
цього вузла
print G.out_degree(1, weight='weight') # сума ваг
вихідних ребер цього вузла
G2=G.reverse() # обернений граф
G2=G.subgraph([1,3,4]) # підграф
# див. також функції union, disjoint_union,
cartesian_product, compose, complement
print G.has_edge(1,2) # чи граф має ребро 1,2
print G.has_node(1) # чи граф має вузол 1
print G.has_predecessor(2,1) # чи вузол 2 має предка
1
print G.has_successor(1,2) # чи вузол 1 має нащадка 2
```

```
print list(G.nodes_with_selfloops()) # вузли з
ребрами, які виходять і входять в цей вузол
print list(G.selfloop_edges()) # ребра з однаковим
вузлом на двох кінцях

nx.draw_spectral(G, with_labels=True) # нарисувати
граф
plt.show(); print "Рисунок - Візуалізація
орієнтованого графа"

# Алгоритми на графах:
print nx.is_tree(G) # чи це дерево?
try: print nx.find_cycle(G) # чи є цикли?
except: print "No cycle found"
print list(nx.dfs_edges(G,1)) # ітерація по ребрам
для пошуку в глибину
print list(nx.bfs_edges(G,1)) # для пошуку в ширину
(починати з 1)
print dict(nx.all_pairs_shortest_path(G)) #
найкоротший шлях між усіма вузлами
print nx.shortest_path(G,1,4) # найкоротший шлях від
1 до 4
print nx.dijkstra_path(G,1,4) # або
print 'PageRank', nx.pagerank(G, alpha=0.9) #
алгоритм ранжування PageRank
print 'HITS', nx.hits(G) # алгоритм ранжування HITS
повертає вузли, які посилаються на авторитетні вузли
і ці авторитетні вузли
```

```
suc [2, 3]
pre []
nei [2, 3]
out [(1, 2), (1, 3)]
in []
2
```

```
1.0
1.0
True
True
True
True
[]
[]
```

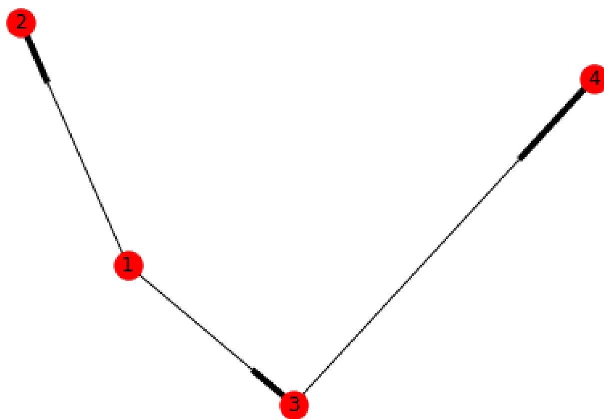


Рисунок 45 - Візуалізація орієнтованого графа

```
True
No cycle found
[(1, 2), (1, 3), (3, 4)]
[(1, 2), (1, 3), (3, 4)]
{1: {1: [1], 2: [1, 2], 3: [1, 3], 4: [1, 3, 4]}, 2:
{2: [2]}, 3: {3: [3], 4: [3, 4]}, 4: {4: [4]}}
[1, 3, 4]
[1, 3, 4]
PageRank {1: 0.16116025883844023, 2: 0.23368249652134
615, 3: 0.23368249652134615, 4: 0.3714747481188674}
```

```
HITS ({1: 0.9999999990686774, 2: 0.0, 3: 9.3132257374
81168e-10, 4: 0.0}, {1: 0.0, 2: 0.4999999990686774, 3
: 0.4999999990686774, 4: 1.86264514576151e-09})
```

pyDatalog - логічне програмування в Python

Логічне програмування основане на виведенні нових фактів з існуючих відповідно правил логічного виведення. pyDatalog 0.17.1 (http://sites.google.com/site/pydatalog) - пакет, який додає парадигму логічного програмуавання в Python. Datalog - повністю декларативна підмножина мови логічного програмування Prolog. В декларативному програмуванні програма описує **що** потрібно досягти, а в імперативному - **як**. Програма мовою Datalog містить факти, правила логічного виведення і запити. Наприклад, фактом є твердження "Іван є батьком Петра", правило логічного виведення - "якщо Y батько X, то X дитина Y", а запит - "знайти усіх дітей Петра".

```
from __future__ import unicode_literals
from pyDatalog import pyDatalog
pyDatalog.create_terms("isParent, isChild, isSibling,
X, Y, Z, c") # Datalog-терми (змінні з великої букви)
+isParent("Ivan","Petro") # додати факт (isParent -
предикат)
+isParent("Ivan","Stepan") # предикати можуть бути
кирилицею: globals()['назва']
# правила логічного виведення ("<=" - "якщо, то"):
isChild(X,Y) <= isParent(Y,X) # якщо Y батько X, то X
дитина Y
isSibling(X,Y) <= isParent(Z,X) & isParent(Z,Y) &
~(X==Y)
# запити:
print isChild("Petro", X).data # знайти батька Петра
print isChild(X,"Ivan").data # знайти усіх дітей
Івана
print isSibling(X,Y).data # знайти усіх братів
```

```
(c[X]==len_(Y)) <= (isParent(X,Y))
print (c[X]==Y).data # знайти кількість дітей батька 
X

pyDatalog.clear() # очистити базу даних
pyDatalog.create_terms("abs, f, g") # abs - вбудована 
функція
print ((X==[1,2,-3]) & (Y==abs(X[2])+1)).data # 
знайти X,Y
print (X.in_(range(5)) & Y.in_(range(5)) & (Z==X+Y) & 
(Z<2)).data # знайти X,Y
f["Ivan"]=2 #  факт (f - предикат)
f["Petro"]=0
#+(f['Petro'] == 0) # або
print ((f[X]==Y) & (Y>0)).data # знайти X,Y
del f["Ivan"] # видалити
(g[X]==3) <= (X=="Ivan")
print ((g[X]==Y)).data # знайти X,Y
```

```
[(u'Ivan',)]
[(u'Petro',), (u'Stepan',)]
[(u'Petro', u'Stepan'), (u'Stepan', u'Petro')]
[(u'Ivan', 2)]
[((1, 2, -3), 4)]
[(0, 1, 1), (1, 0, 1), (0, 0, 0)]
[(u'Ivan', 2)]
[(u'Ivan', 3)]
```

Зв’язок з інтерпретатором Prolog

В прикладі показано спосіб взаємодії Python програми з інтерпретатором Prolog (SWI-Prolog) за допомогою `subprocess.Popen`. Більш тісний зв’язок з SWI-Prolog реалізує пакет PySwip (http://github.com/yuce/pyswip).

```
from subprocess import Popen, PIPE, STDOUT
with open('family.pl', 'w') as f: # створити Prolog-програму
    f.write('isParent("Ivan","Petro"). isChild(X,Y) :- isParent(Y,X).')
p = Popen(r'"c:\Program Files (x86)\swipl\bin\swipl.exe" -q family.pl', shell=True, stdin=PIPE, stdout=PIPE, stderr=STDOUT)
print p.communicate('isChild(X,"Ivan").') # надіслати запит і отримати результат
```

```
('\r\nX = "Petro".\r\n\r\n\r\n', None)
```

kanren - логічне програмування в Python

kanren 0.2.3 (https://github.com/logpy/logpy) - це реалізація вбудованої предметно-орієнтованої мови логічного програмування miniKanren (http://minikanren.org). miniKanren спроектована для легкої модифікації і розширення для різних видів логічного програмування. kanren використовує бібліотеку `unification` для уніфікації - розширеної форми зіставлення з взірцем. Типи, які можуть бути уніфіковані, можуть застосовуватись також для логічного програмування. Наступний код знаходить значення `x`:

```
>>> from unification import *
>>> x = var('x')
>>> unify((1,x,3), (1,2,3))
{~x: 2}
```

Де `(1,x,3)`, `(1,2,3)` - два терми (дерево, листки якого є константами або змінними), `x` - логічна змінна, `{~x: 2}` - підстановка, `unify` - функція, яка повертає підстановку за двома термами.

Ціль - це функція з підстановки в потік підстановок:

```
>>> goal=eq((1,x,3), (1,2,3))
>>> for s in goal({}):
```

```
...     print s
{~x: 2}
```

Ціль створюється за допомогою конструктора цілі (`eq`, `membero`, `conde` та інших). Логічна програма виконується функцією `run(n, x, *goals)`, де `n` - кількість розв'язків, `x` - змінна, `*goals` - послідовність цілей.

```
from kanren import *
ne = goals.not_equalo # конструктор цілі "не рівні"
#ne = goalify(lambda x,y: x!=y) # або створити з
функції
#import operator; ne = goalify(operator.ne) # або
створити з оператора

parent = Relation() # відношення між термами
fact(parent, "Іван", "Петро") # факт - Іван батько
Петра
fact(parent, "Іван", "Василь")
fact(parent, "Петро",  "Марія")
x,y = var(),var() # логічні змінні
def sibling(x, y): # відношення "рідні" (брат або
сестра)
    z = var()
    # (z батько x) і (z батько y) і (x,y не рівні)
    # conde - конструктор цілі для логічного І та АБО
    return conde( (parent(z, x), parent(z, y),
(ne,(x,y),True)) )

for x,y in run(0, (x, y), sibling(x, y)):
    print x,y # вивести усі пари рідних
x,y = var(),var()
for x in run(0, x, conde( (parent(x, "Петро"),),
(parent("Петро", x),) )):
    print x # вивести усіх батьків або дітей Петра
```

```
Петро Василь
Василь Петро
Іван
Марія
```

python-constraint - задачі виконання обмежень

python-constraint 1.3.1 (http://pypi.org/project/python-constraint) - модуль для розв'язування задач виконання обмежень, ціллю яких є знаходження значень змінних, які відповідають заданим обмеженням. Щоб сформулювати таку задачу потрібно визначити змінні, множину їх значень і обмеження. Модуль може бути використаний для програмування в обмеженнях, яке є видом декларативного програмування. В модулі доступні такі види обмежень: `FunctionConstraint`, `AllDifferentConstraint`, `AllEqualConstraint`, `ExactSumConstraint`, `MaxSumConstraint`, `MinSumConstraint`, `InSetConstraint`, `NotInSetConstraint`, `SomeInSetConstraint`, `SomeNotInSetConstraint`.

```
from constraint import *
problem = Problem() # створити задачу
problem.addVariable('a', [1,2,3]) # додати змінну і
множину її значень
problem.addVariable('b', [1,2,4])
print problem.getSolutions() # розв'язати задачу
(обмежень немає)
# додати обмеження (розкоментуйте потрібні):
problem.addConstraint(lambda a,b: a+b>3, ('a', 'b'))
# a+b>3
#problem.addConstraint(AllDifferentConstraint()) # a
і b різні
#problem.addConstraint(AllEqualConstraint()) # a і b
однакові
#problem.addConstraint(InSetConstraint([2,3])) # a і
b в множині {2,3}
```

```
print problem.getSolutions() # розв'язати задачу -
знайти значення a і b, які відповідають обмеженням
```

```
[{'a': 3, 'b': 4}, {'a': 3, 'b': 2}, {'a': 3, 'b': 1}
, {'a': 2, 'b': 4}, {'a': 2, 'b': 2}, {'a': 2, 'b': 1
}, {'a': 1, 'b': 4}, {'a': 1, 'b': 2}, {'a': 1, 'b':
1}]
[{'a': 3, 'b': 4}, {'a': 3, 'b': 2}, {'a': 3, 'b': 1}
, {'a': 2, 'b': 4}, {'a': 2, 'b': 2}, {'a': 1, 'b': 4
}]
```

PIL (Pillow) - робота з растровою графікою

Pillow 4.2.1 (Python Imaging Library) - це бібліотека для роботи з растровою графікою (http://pillow.readthedocs.io). Підтримує велику кількість форматів, їх конвертацію, різні операції з зображенням.

```
from PIL import Image, ImageDraw, ImageFont,
ImageFilter
image = Image.new('RGBA', (50, 40), (0, 0, 0, 0)) #
зображення з заданою колірною моделлю, розміром і
фоном
draw = ImageDraw.Draw(image) # простий інтерфейс для
2D рисування
fnt =
ImageFont.truetype(r'c:\Windows\Fonts\times.ttf', 28)
# шрифт
draw.text((5, 5), "PIL", font=fnt,
fill=(0,255,0,255)) # рисувати текст в заданих
координатах
image2=image.rotate(20) # повернути на 20 градусів
image2=image2.filter(ImageFilter.SMOOTH) # згладити
зображення
image=Image.alpha_composite(image,image2) # об'єднати
image = image.crop([1, 1, 49, 39]) # обрізати
```

```
image.convert('RGB') # конвертувати в модель RGB
#image.save("pil.png") # зберегти
image.show() # показати у зовнішній програмі
```



Рисунок 46 - Растровий рисунок

PyOpenGL - прив’язка до OpenGL

OpenGL - це незалежний від мови програмування і платформи API для рендерингу 2D і 3D векторної графіки. Найчастіше використовується з графічним процесором в програмах для візуалізації, САПР, іграх. PyOpenGL 3.1.0 (http://pyopengl.sourceforge.net) - це прив’язка Python до OpenGL v1.1-4.4, яка створена за допомогою ctypes. Підтримує багато GUI-бібліотек та пов’язаних з OpenGL бібліотек (GLES, GLU, EGL, WGL, GLX, FreeGLUT, GLE). FreeGLUT (http://freeglut.sourceforge.net) - це бібліотека, яка призначена для таких системних задач, як створення вікон, ініціалізація контексту OpenGL і обробка подій. Для роботи програми знадобиться freeglut 3.0.0 MSVC Package (http://www.transmissionzero.co.uk/software/freeglut-devel). Для вивчення PyOpenGL може бути використана офіційна документація (http://www.opengl.org/sdk/docs/man2) та (http://freeglut.sourceforge.net/docs/api.php). Існує також інша прив’язка до OpenGL, яка є частиною pyglet.

```
from OpenGL.GLUT import *
from OpenGL.GL import *
from OpenGL.GLU import *
T=[0,0,0] # вектор переміщень
R=[0,0,1,0] # вектор повороту
def display(  ): # функція відображення OpenGL - рисує об'єкти
```

```
    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT) # очистити буфери кольору і
глибини
    glColor3f(0, 0, 0) # установити колір RGB
(чорний)
    glMatrixMode(GL_MODELVIEW) # режим матриці
вигляду
    glLoadIdentity() # одинична матриця
    glTranslatef(T[0], T[1], T[2]) # множить поточну
матрицю на матрицю переміщення
    glRotatef(R[0], R[1], R[2], R[3]) # множить
поточну матрицю на матрицю повороту навколо вектора

    # створює кольорові трикутники
    glBegin(GL_TRIANGLE_STRIP) # розмежовує вершини
примітива (дозволено GL_POINTS, GL_LINES,
GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES,
GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS,
GL_QUAD_STRIP, GL_POLYGON)
    glVertex3f(0.5, 0.5, 0.5) # перша вершина
трикутника
    glColor3f(0.9, 0.9, 0.9) # колір наступних вершин
    glVertex3f(-0.5, -0.5, 0) # друга вершина
    glColor3f(0.1, 0.1, 0.1) # колір наступних вершин
    glVertex3f(0.5, -0.5, 0) # третя вершина
    glVertex3f(0.5, 0.5, -0.5) # вершина другого
трикутника
    glEnd() # завершити список вершин примітива

    glPointSize(3) # розмір точки
    glBegin(GL_POINTS) # точка
    glVertex3f(0,0,0)
    glEnd()

    glBegin(GL_LINES) # рисуємо 3 лінії - осі
```

```
координат X,Y,Z
    p1=0,0,0
    # для кожного кольору і другої точки лінії
    for c,p2 in [[(1, 0, 0),(1, 0, 0)],[(0, 1, 0),(0,
1, 0)],[(0, 0, 1),(0, 0, 1)]]:
        glColor3f(*c)
        glVertex3f(*p1)
        glVertex3f(*p2)
    glEnd() # завершити рисування

    glutWireCube(1) # нарисувати куб

    glLineWidth(2) # ширина ліній
    glPushMatrix() # запам'ятати глобальну систему
координат
    glTranslatef(0, 0.5, 0) # перемістити систему
координат вздовж Y
    glRotatef(45, 0, 0, 1) # повернути в новій
системі координат навколо осі Z
    # спробуйте поміняти дві попередні команди
місцями
    glutWireCube(0.5) # нарисувати куб
    glPopMatrix() # відновити глобальну систему
координат

    glTranslatef(0, -0.5, 0) # перемістити систему
координат вздовж Y
    # спробуйте закоментувати попередні команди
glPushMatrix і glPopMatrix
    glutWireCube(0.25) # нарисувати куб

    #glFlush() # виконати GL команди
    glutSwapBuffers() # переключити буфери в режимі
подвійної буферизації
```

```
def specialKeyPressed(key, x, y): # переміщує або
повертає, якщо натиснуті спеціальні клавіші
    global T,R
    if key == GLUT_KEY_LEFT: T[0] -= 0.1
    elif key == GLUT_KEY_RIGHT: T[0] += 0.1
    elif key == GLUT_KEY_DOWN: T[1] -= 0.1
    elif key == GLUT_KEY_UP: T[1] += 0.1
    elif key == GLUT_KEY_PAGE_DOWN: R[0] += 5
    elif key == GLUT_KEY_PAGE_UP: R[0] += -5
    glutPostRedisplay()

glutInit() # функція ініціалізації glut
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB) # режим
відображення
glutInitWindowSize(250, 250) # розмір вікна
glutInitWindowPosition(100, 100) # позиція вікна
glutCreateWindow("My PyOpenGL Demo") # створити вікно
glutSpecialFunc(specialKeyPressed)
glClearColor(255, 255, 255, 0) # визначає RGBA колір,
який буде використовувати glClear
glShadeModel(GL_SMOOTH) # модель затінення GL_FLAT
або GL_SMOOTH

# параметри матеріалів і освітлення (розкоментуйте
щоб задіяти)
'''
glMaterialfv(GL_FRONT, GL_AMBIENT, [0.2, 0.2, 0.2,
1.0])
glMaterialfv(GL_FRONT, GL_DIFFUSE, [0.8, 0.8, 0.8,
1.0])
glMaterialfv(GL_FRONT, GL_SPECULAR, [1.0, 0.0, 1.0,
1.0])
glMaterialfv(GL_FRONT, GL_SHININESS, 50.0)
glLightfv(GL_LIGHT0, GL_AMBIENT, [0.0, 1.0, 0.0,
1.0])
```

```
glLightfv(GL_LIGHT0, GL_DIFFUSE, [1.0, 1.0, 1.0,
1.0])
glLightfv(GL_LIGHT0, GL_SPECULAR, [1.0, 1.0, 1.0,
1.0])
glLightfv(GL_LIGHT0, GL_POSITION, [1.0, 1.0, 1.0,
0.0]);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, [0.2, 0.2,
0.2, 1.0])
glEnable(GL_LIGHTING)
glEnable(GL_LIGHT0)
glDepthFunc(GL_LESS)
'''
glEnable(GL_DEPTH_TEST) # активізувати перевірку
глибини (не показувати невидимі поверхні)

glutDisplayFunc(display) # вказати функцію
відображення
glutMainLoop() # головний цикл програми
```

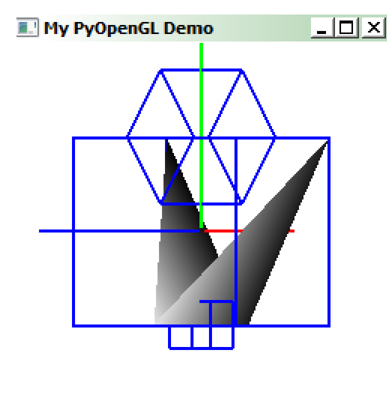


Рисунок 47 - Вікно програми

pyglet - кросплатформна віконна і мультимедійна бібліотека

pyglet 1.3.2 (http://bitbucket.org/pyglet/pyglet) - це віконна і мультимедійна бібліотека, яка призначена для розробки ігор та інших візуально багатих програм. Підтримує обробку вікон, обробку подій GUI, графіку OpenGL, завантаження зображень і відео, а також відтворення звуків и музики. Працює на Windows, OS X і Linux. В прикладі показано простий переглядач 3D об’єктів OpenGL на основі pyglet.

```
import pyglet
from pyglet.window import Window, mouse
from pyglet.gl import *

class MyWindow(Window): # клас вікна
    def __init__(self, *args, **kwargs):
        super(MyWindow,self).__init__(*args,
**kwargs)
        self.x,self.y = 0,0 # кути повороту
        self.label =
pyglet.text.Label(x=20,y=20,color=(0, 0, 0, 255)) #
надпис
        glClearColor(1, 1, 1, 1) # визначає RGBA
колір, який буде використовувати glClear
        glEnable(GL_DEPTH_TEST) # активізувати
перевірку глибини (не показувати невидимі поверхні)

    def on_resize(self, width, height): # під час
зміни розміру вікна
        aspectRatio = width/height # відношення
сторін
        glViewport(0, 0, width, height) #
установлення порту виведення
        glMatrixMode(GL_PROJECTION) # режим матриці
проекцій
        glLoadIdentity() # одинична матриця
```

```
        gluPerspective(35.0, aspectRatio, 1.0,
1000.0) # матриця перспективної проекції
        glMatrixMode(GL_MODELVIEW) # режим матриці
вигляду
        glLoadIdentity()

    def on_draw(self): # під час необхідності
перерисування

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT) #
очистити буфери кольору і глибини
        glMatrixMode(GL_MODELVIEW) # режим матриці
вигляду
        glPushMatrix() # запам'ятати глобальну
систему координат
        glTranslatef(0, 0, -600) # перемістити
систему координат вздовж Z
        self.label.text='ax=%d ay=%d'%(self.x,self.y)
# змінити надпис
        self.label.draw() # нарисувати надпис
        glRotatef(self.x, 1, 0, 0) # повернути в
новій системі координат навколо осі X
        glRotatef(self.y, 0, 1, 0) # повернути в
новій системі координат навколо осі Y
        self.drawAxes() # нарисувати осі
        self.drawObject() # нарисувати об'єкт
        glPopMatrix() # відновити глобальну систему
координат

    def drawAxes(self): # рисує осі X,Y,Z
        glBegin(GL_LINES) # розмежовує вершини
примітива (лінії)
        for r,g,b,x,y,z in [(1,0,0,1000,0,0),
(0,1,0,0,1000,0), (0,0,1,0,0,1000)]:
            glColor3f(r, g, b) # колір наступної
```

```
вершини
            glVertex3f(-x, -y, -z) # перша вершина
            glVertex3f(x, y, z) # друга вершина
        glEnd() # завершити список вершин примітива

    def drawObject(self): # рисує об'єкт
        glBegin(GL_TRIANGLES) # розмежовує вершини
примітива (трикутника)
        glColor3f(0, 0, 1) # колір наступної вершини
        glVertex3f(0, 0, 0) # перша вершина
        glVertex3f(100, 0, 0) # друга вершина
        glVertex3f(100, 100, 0) # третя вершина
        glEnd() # завершити список вершин примітива

    def on_mouse_drag(self, x, y, dx, dy, buttons,
modifiers): # під час руху миші
        if buttons & mouse.LEFT:
            self.x-=dy # змінити кут повороту на
величину переміщення миші
            self.y+=dx

MyWindow(width=400, height=400,
caption="pyglet",resizable=True) # вікно
pyglet.app.run() # цикл обробки подій
```

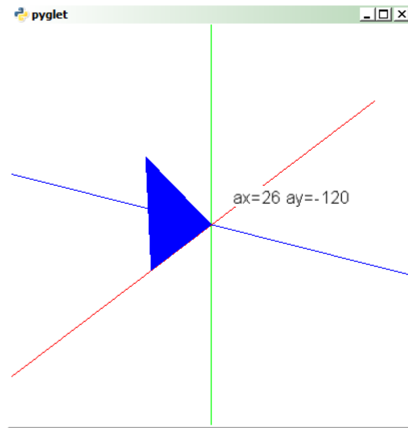


Рисунок 48 - Вікно програми

pythonOCC - прив'язка до геометричного ядра Open CASCADE Technology

Open CASCADE Technology (OCCT) - вільна бібліотека мовою C++ для геометричного моделювання (геометричне ядро), яка найчастіше використовується для створення САПР. Геометричні моделі створюються способом граничного подання BREP (boundary representation) у вигляді топологічних форм (вершин, ребер, циклів, граней, поверхонь, твердотільних форм і їх поєднань). pythonOCC 0.18.1 (http://www.pythonocc.org) - бібліотека Python, що дозволяє використання класів OCCT [34]. Побудована за допомогою SWIG - інструмента для пов'язування коду C++ та Python.

```
from math import pi
from OCC.gp import * # геометричний процесор gp - незбережувані базові геометричні об'єкти
#from OCC.Geom import * # збережувані базові 3D геометричні об'єкти
from OCC.GC import * # алгоритми для побудови елементарних геометричних об'єктів OCC.Geom
from OCC.BRepBuilderAPI import * # класи для
```

```
моделювання і побудови чисто топологічних структур
даних

p1=gp_Pnt(1, 0, 0) # точка (gp_Pnt)
p2=gp_Pnt(1, 2, 0) # точка
p3=gp_Pnt(2, 1, 0) # точка
edge1=BRepBuilderAPI_MakeEdge(p1,p2).Edge() # ребро
(TopoDS_Edge)
arc=GC_MakeArcOfCircle(p1,p3,p2).Value() # дуга
(Handle_Geom_TrimmedCurve)
edge2=BRepBuilderAPI_MakeEdge(arc).Edge() # створює
ребро з дуги

mw=BRepBuilderAPI_MakeWire() # створює контур
(BRepBuilderAPI_MakeWire)
mw.Add(edge1) # додати ребро
mw.Add(edge2) # додати ребро
wire=mw.Wire() # контур (TopoDS_Wire)

face=BRepBuilderAPI_MakeFace(wire).Face() # грань
(TopoDS_Face)
vector=gp_Vec(p1, gp_Pnt(1, 0, 1)) # вектор (gp_Vec)
from OCC.BRepPrimAPI import * # забезпечує API для
створення примітивів (призм, тіл обертання,
витягувань, сфер, циліндрів ...)
solid1 = BRepPrimAPI_MakePrism(face, vector).Shape()
# призма (TopoDS_Shape)

axis=gp_Ax1(gp_Pnt(),gp_Dir(0,1,0)) # вісь Y (gp_Ax1)
solid2 = BRepPrimAPI_MakeRevol(face, axis,
pi).Shape() # тіло обертання (TopoDS_Shape)

from OCC.BRepAlgoAPI import * # забезпечує новий API
для булевих операцій з формами (об'єднань, вирізів,
перетинів)
```

```
solid3=BRepAlgoAPI_Fuse(solid2, solid1).Shape() #
тіло після об'єднання (TopoDS_Shape); деколи важлива
послідовність аргументів

from OCC.Display.SimpleGui import init_display #
засоби для створення GUI
display, start_display, add_menu,
add_function_to_menu = init_display()
display.set_bg_gradient_color(255,255,255,255,255,255
) # колір фону
display.DisplayShape(solid3) # показати форму
display.FitAll()
start_display()
```

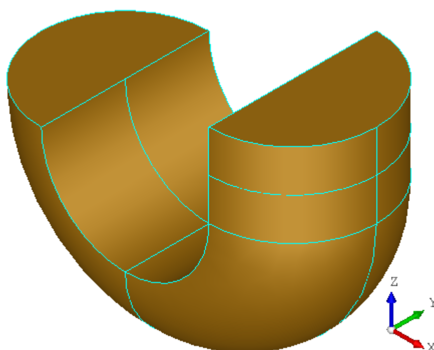


Рисунок 49 - Результати виконання програми

FreeCAD - вільна САПР з Python API

FreeCAD 0.17 (http://www.freecadweb.org) - це вільна параметрична 3D САПР, яка базується на геометричному ядрі Open CASCADE Technology 7.2.0 і володіє Python API. Геометричні моделі створюються способом граничного подання BREP за допомогою її Python-модуля Part, який є прямим зв’язком з OCCT [34]. Повністю OCCT доступна з PythonOCC, але використання

FreeCAD модуля Part набагато зручніше. Для виконання прикладу введіть в консолі:

```
"e:\FreeCAD 0.17x64\bin\python.exe" main.py
```

Для виконання з довільного інтерпретатора Python27x64 введіть в консолі:

```
c:\Python27\python.exe main.py
```

```
import sys
FREECADPATH = r"e:\FreeCAD 0.17x64\bin"
sys.path.append(FREECADPATH) # шлях до бібліотек
FreeCAD

import math
import FreeCAD as App # модуль для роботи з програмою
import FreeCADGui as Gui # модуль для роботи з GUI
import Part # workbench-модуль для створення і
керування BRep об'єктами

v1=App.Vector(0,0,0) # вектор (або точка)
v2=App.Vector(0,10,0)
v3=App.Vector(5,5,0)
l1=Part.LineSegment(v1,v2) # лінія
e1=l1.toShape() # ребро
# або e1=Part.makeLine((0,0,0),(0,10,0)) # ребро
a1=Part.Arc(v1,v3,v2) # дуга за трьома точками
e2=a1.toShape() # ребро
# або
e2=Part.makeCircle(5,App.Vector(0,5,0),App.Vector(0,0
,1),-90,90)

bs=Part.BSplineCurve() # B-сплайн
bs.interpolate([(0,0,0),(0,1,1),(0,-1,2)]) # шляхом
інтерполяції
```

```
# або
#bs.approximate([(0,0,0),(0,1,1),(0,-1,2)]) # шляхом
апроксимації
#bs.buildFromPoles([(0,0,0),(0,1,1),(0,-1,2)]) # за
списком полюсів
e3=bs.toShape() # ребро

w1=Part.Wire([e1,e2]) # цикл (сукупність ребер)
f1=Part.Face(w1) # грань
trsf=App.Matrix() # матриця трансформації
trsf.rotateZ(math.pi/4) # повернути навколо осі z
trsf.move(App.Vector(5,0,0)) # перемістити
f2=f1.copy() # копія форми
f2.transformShape(trsf) # виконати трансформацію
# або
#
f2.rotate(App.Vector(0,0,0),App.Vector(0,0,1),180.0/4
)
# f2.translate(App.Vector(5,0,0))
s1=f2.extrude(App.Vector(0,0,10)) # тіло шляхом
видавлювання
s2=Part.Wire([e3]).makePipe(f1) # тіло шляхом
видавлювання по траєкторії
# або
s2=Part.Wire([e3]).makePipeShell([w1],True,True)
s3=f1.revolve(v1,App.Vector(0,1,0),90) # тіло шляхом
обертання
s2=s2.fuse(s3) # об'єднання тіл (див. також common,
cut, oldFuse)
s2=s2.removeSplitter() # видалити непотрібні ребра
(refine shape)
# див. також makeBox, makeCylinder, makeLoft,
makeThickness, ...
s1=s1.makeFillet(1,[s1.Edges[1]]) # скруглення (див.
також makeChamfer)
```

```
print s1.ShapeType # тип форми
print s1.Volume # об'єм (див. також Length, Area,
CenterOfMass)
print s1.distToShape(s2)[0] # мінімальна відстань до
іншої форми
print s1.Faces[0] # перша грань
print s1.Edges[0] # перше ребро
print type(s1.Edges[0].Curve) # тип кривої першого
ребра
print s1.Vertexes[0].Point.x # координата x точки
першої вершини

#s1.exportBrep("my.brep") # експорт у форматі BREP
(див. також exportStep, exportIges)
#s1 = Part.Shape()
#s1.read("my.brep") # імпорт у форматі BREP

# Наступні команди потрібні тільки для візуалізації
створених форм
Gui.showMainWindow() # показати головне вікно
doc=App.newDocument() # створити новий документ
for shape in [l1.toShape(), a1.toShape(), w1, f1, f2,
s1, bs.toShape(), s2]:
    Part.show(shape) # показати форму
doc.recompute() # перебудувати
Gui.exec_loop() # головний цикл програми
```

```
Compound
389.596349447
0.0
<Face object at 000000003AF38D0>
<Edge object at 000000003AF4350>
<type 'Part.Line'>
-1.27414669346
```

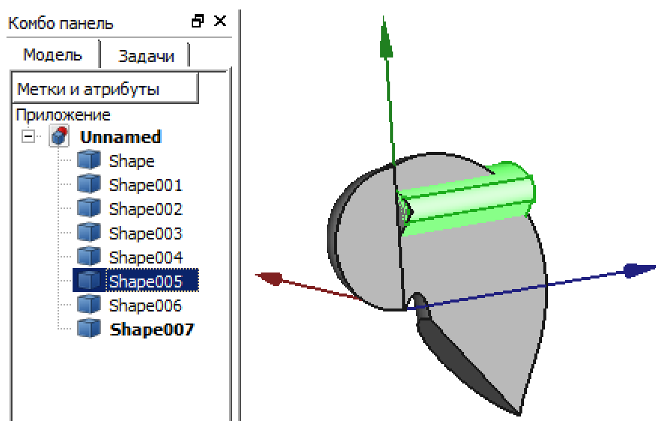


Рисунок 50 - Результати виконання програми

Abaqus/CAE - моделювання методом скінченних елементів

Abaqus/CAE 6.14 (http://www.3ds.com/products-services/simulia) - комерційне середовище для розв'язування задач механіки деформівного твердого тіла, гідрогазодинаміки і електродинаміки методом скінченних елементів (МСЕ). Володіє зручним API мовою Python 2.7, який дозволяє створювати прикладні програми. В прикладі створюється осесиметрична модель деталі, яка розтягується осьовим навантаженням. Зазвичай послідовність розв'язування задач МСЕ містить етапи: створення геометрії, властивостей матеріалу, генерація сітки елементів, створення граничних умов, розв'язування рівнянь і аналіз результатів.

```
from abaqus import *
from abaqusConstants import *
Mdb() # створити нову модель
m=mdb.models['Model-1'] # модель
import os
os.chdir(r"C:\Abaqus") # робочий каталог

s=m.ConstrainedSketch(name='__profile__',
```

```
sheetSize=200.0) # ескіз
s.ConstructionLine((0.0, -100.0), (0.0, 100.0)) #
допоміжна лінія
points=[(10.0, 0.0),(0.0, 0.0),(0.0, 10.0),(10.0,
10.0),(5.0, 5.0), (10.0, 0.0)] # точки ескізу
for p1,p2 in zip(points[:-1],points[1:]):
    s.Line(p1,p2) # лінія за точками
p=m.Part(name='Part-1', dimensionality=AXISYMMETRIC,
type=DEFORMABLE_BODY) # деталь
p.BaseShell(sketch=s) # на основі ескізу s

mat=m.Material(name='Material-1') # матеріал
mat.Elastic(table=((2.1e11, 0.3), )) # модуль Юнга і
коеф. Пуасона
m.HomogeneousSolidSection(name='Section-1',
material='Material-1', thickness=None) # однорідна
секція матеріалу
region = p.Set(faces=p.faces, name='Set-1') #
геометричний регіон
p.SectionAssignment(region=region,
sectionName='Section-1') # зв'язати секцію і деталь

a = m.rootAssembly # збірка
inst=a.Instance(name='Part-1-1', part=p,
dependent=ON) # її елемент
a.Set(vertices=inst.vertices.findAt(((5,5,0),)),
name='CenterPoint') # точка для результатів
p.seedPart(size=1.0, deviationFactor=0.1,
minSizeFactor=0.1) # розміри сітки
p.generateMesh() # створити сітку

m.StaticStep(name='Step-1', previous='Initial') #
статичний крок
region=a.Set(edges=inst.edges.findAt(((1,0,0),)),
name='Encastre')
```

```
m.EncastreBC(name='BC-1', createStepName='Step-1',
region=region, localCsys=None) # гранична умова на
нижньому торці
region=a.Surface(side1Edges=inst.edges.findAt(((1,10,
0),)), name='Surf-1')
m.Pressure(name='Load-1', createStepName='Step-1',
region=region,  magnitude=-10e6) # тиск на верхньому
торці

job=mdb.Job(name='Job-1', model='Model-1') # створити
задачу
job.submit() # надіслати розв'язувачу
job.waitForCompletion() # чекати завершення

from visualization import * # для візуалізації
результатів
odb=openOdb("C:/Abaqus/Job-1.odb") # відкрити базу
даних результатів
f=odb.steps['Step-1'].frames[-1].fieldOutputs #
результати останнього фрейму
reg=odb.rootAssembly.elementSets['ENCASTRE'] # нижній
торець
print f['S'].getSubset(position=INTEGRATION_POINT,
region=reg).values[0].mises # еквівалентне напруження
в елементі 0 регіону
reg=odb.rootAssembly.nodeSets['CENTERPOINT'] #
центральна точка
print f['U'].getSubset(position=NODAL,
region=reg).values[0].data # переміщення Ux і Uy
(.magnitude - сумарне)
odb.close() # закрити базу даних результатів
```

```
7578833.5
[ 0.00022752  0.00070781]
```

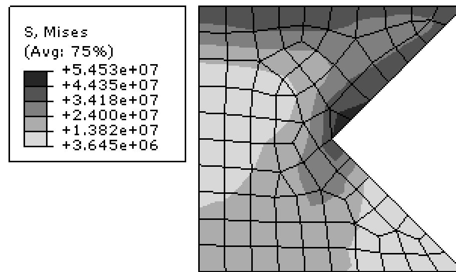


Рисунок 51 - Еквівалентні напруження за критерієм Мізеса-Губера (Па)

SymPy - символьна математика

SymPy (http://www.sympy.org) - це бібліотека для символьної математики, яка призначена для роботи з математичними виразами в аналітичній (символьній) формі на відміну від чисельних обчислень в SciPy. Її можна розглядати як вільну альтернативу системам символьної математики Maple, Mathcad, Mathematica. Для прикладу SymPy дозволяє інтегрувати, диференціювати, спрощувати вирази, розв'язувати рівняння в символьній формі.

```
from sympy import *
x,y,lamda=symbols('x y lamda') # визначити змінні
expr=x+y # вираз
expr2=expr+x # вираз
print expr2.subs(y,2) # вираз шляхом підстановки y=2
print srepr(expr2) # низькорівневе представлення
виразу
print expr2.args # кортеж усіх складових виразу
print expr2.atoms() # атоми виразу
print expr2.atoms(Symbol) # атоми (типу Symbol)
виразу
print expr2.subs([(x,5),(y,2)]).evalf() # підставити
у вираз і обчислити
#print expr2.subs({x:5, y:2}).evalf() # або так
```

```
#print expr2.evalf(subs={x:5, y:2}) # або так
#print N(expr2, subs={x:5, y:2}) # або так
print sympify("x**2-1/2") # перетворити рядок у вираз
SymPy
expr3=sin(x) # вираз
f=lambdify(x, expr3,"math") # функція для швидкого
розрахунку числових значень. Третім аргументом може
бути "math" або "numpy" або, наприклад, {"sin":mysin}
print f(0.1) # числове значення

expr4=Integral(sqrt(1/x), x) # вираз-інтеграл
pprint(expr4, use_unicode=False) # виведення в
Unicode (True) або ASCII (False). Для Unicode
використовуйте для виведення IPython QTConsole або
IPython notebook
print latex(expr4) # вивести як LaTeX
from sympy.printing.mathml import *
print mathml(expr4) # вивести як MathML

print simplify(sin(x)**2 + cos(x)**2) # спростити
вираз
print expand((x + 1)**2) # розширити поліноміальний
вираз

print diff(x**2, x) # похідна
print diff(x**2, x,x) # похідна другого порядку
#print diff(x**2, x,2) # або так
print expr3.diff(x) # похідна
expr5=Derivative(x**2,x) # нерозрахована похідна
init_printing(use_unicode=False) # не використовувати
Unicode для виведення
pprint(expr5) # вивести вираз в математичному вигляді
print expr5.doit() # розрахувати похідну
print diff(exp(x*y),x,x,y,y) # мішана похідна
print integrate(sin(x),x) # невизначений інтеграл
```

```
print integrate(exp(-x), (x, 0, oo)) # визначений
інтеграл
print integrate(exp(-x**2 - y**2), (x, -oo, oo), (y,
-oo, oo))
print limit(sin(x)/x, x, 0) # границя
print series(sin(x), x, 0, 8) # ряд функції

print solve(Eq(x**2, 4), x) # розв'язати рівняння
#print solve(x**2-4, x) # або так
# розв'язку не знайдено, якщо solve повертає [] або
викликає NotImplementedError
print solve([x-y, x+y], [x, y], dict=True) #
розв'язати систему рівнянь
#plot((x-2)**2-2) # нарисувати графік функції
z= Symbol('z', real=True, positive=True) # визначити
змінну
print solve([z>2,(z-2)**2-2], z) # розв'язати систему

f=symbols('f', cls=Function) # визначити функцію
diffeq = Eq(f(x).diff(x, x) - 2*f(x).diff(x) + f(x),
0) # диференціальне рівняння
print dsolve(diffeq, f(x)) # розв'язати
диференціальне рівняння
```

```
2*x + 2
Add(Mul(Integer(2), Symbol('x')), Symbol('y'))
(y, 2*x)
set([2, x, y])
set([x, y])
12.0000000000000
x**2 - 1/2
0.0998334166468
  /
 |
 |      ___
```

```
 |    / 1
 |   /  -  dx
 | \/   x
 |
/
\int \sqrt{\frac{1}{x}}\, dx
<apply><int/><bvar><ci>x</ci></bvar><apply><root/><ap
ply><power/><ci>x</ci><cn>-1</cn></apply></apply></ap
ply>
1
x**2 + 2*x + 1
2*x
2
cos(x)
d / 2\
--\x /
dx
2*x
(x**2*y**2 + 4*x*y + 2)*exp(x*y)
-cos(x)
1
pi
1
x - x**3/6 + x**5/120 - x**7/5040 + O(x**8)
[-2, 2]
[{x: 0, y: 0}]
Eq(z, sqrt(2) + 2)
Eq(f(x), (C1 + C2*x)*exp(x))
```

Взаємодія з Maple

Maple (http://www.maplesoft.com) - система комп’ютерної математики з можливостями символьних обчислень. У прикладі показано спосіб взаємодії Python з Maple шляхом створення файлу `mymaple.mpl` з командами Maple та командного файлу

`mymaple.bat`, який створює процес `cmaple.exe`, що виконує ці команди і повертає файл результатів `result.txt`.

```
import os
f=open("d:/mymaple.mpl", "w") # відкрити файл з
командами Maple для запису
f.write(r"evalf(sin(Pi/3));") # записати в файл
команду Maple
f.close() # закрити файл
f=open("d:/mymaple.bat", "w") # відкрити командний
файл для запису
f.writelines((r"path d:\Program Files\Maple
14\bin.win"+"\n",
              r"cmaple.exe < d:\mymaple.mpl >
d:\result.txt"+"\n",
              r"exit")) # записати команди в
командний файл
f.close() # закрити файл
print os.system(r'start /WAIT d:\mymaple.bat') #
виконати команду ОС і чекати її завершення
os.remove(r"d:\mymaple.bat") # видалити файл
os.remove(r"d:\mymaple.mpl")
```

OMPython - інтерфейс OpenModelica Python

Modelica - це основана на рівняннях об'єктно-орієнтована мова для зручного моделювання складних фізичних систем, які містять, наприклад, механічні, електричні, гідравлічні, термічні субкомпоненти. OpenModelica 1.12 (http://www.openmodelica.org/) - це вільне середовище симуляції мовою Modelica. OMPython - це інтерфейс з OpenModelica мовою Python, який забезпечує доступ до OpenModelica API. Для його інсталяції введіть в консолі:

```
cd e:\OpenModelica\share\omc\scripts\PythonInterface
c:\Python27\python.exe -m pip install .
```

В прикладі розв'язується просте диференціальне рівняння з початковою умовою $x(0) = 1$:

$$\frac{dx}{dt} = ax.$$

```
code='''model Simple
    Real x(start=1);
    parameter Real a=1;
equation
    der(x)=a*x;
end Simple;''' # модель мовою Modelica
with open('Simple.mo', 'w') as f: f.write(code) #
створити файл моделі
import os, sys
sys.path.insert(0,
r"e:\OpenModelica\share\omc\scripts\PythonInterface")
# шлях до модулів
from OMPython import OMCSession, ModelicaSystem

# перший спосіб - використання OMCSession:
omc = OMCSession()
omc.sendExpression('loadFile("Simple.mo")')
omc.sendExpression('setParameterValue(Simple, a, 2)')
omc.sendExpression('simulate(Simple)')
omc.sendExpression('plot(x)')
print omc.sendExpression('val(x , 1.0)') # результат
x(time=1.0)

# або більш зручний спосіб:
mod=ModelicaSystem("Simple.mo","Simple")
print mod.getParameters()
mod.setParameters(a=2)
```

```
mod.setSimulationOptions(stopTime=2.0)
mod.simulate()
print mod.getSolutions('time','x') # результати як
масиви

# або компілювати модель і симулювати без OMPython:
omc.sendExpression('buildModel(Simple)') #
компілювати
os.environ["PATH"] += os.pathsep +
r"e:\OpenModelica\bin" # шлях до dll
param='''outputFormat=csv
stopTime=2
a={}
''' # значення параметрів
for i,p in enumerate([1, 2, 3]): # для кожного
значення
    with open('override%d.txt'%i, 'w') as f:
f.write(param.format(p)) # створити файл зі
значеннями параметрів
    os.system(r'Simple.exe -
overrideFile=override%d.txt -r=Simple_%d.csv'%(i,i))
# симуляція
```

```
7.38908993012
{'a': 1.0}
(array([0., 0.002, 0.004, ..., 1.998, 2., 2.]), array
([1., 1.00400801, 1.0080321, ..., 54.38076352, 54.598
72293, 54.59872293]))
```

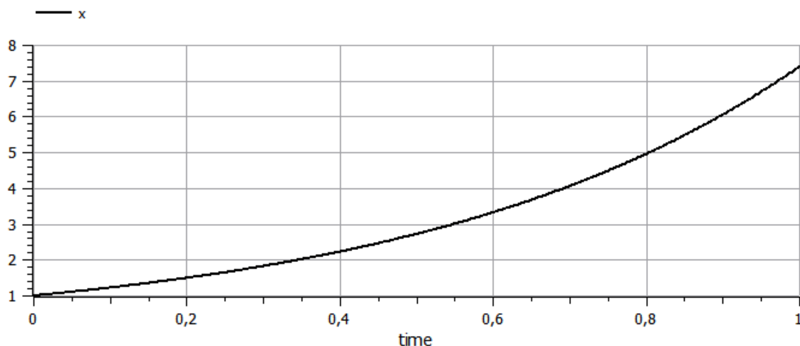


Рисунок 52 - Результати симуляції

xlwt - створення електронних таблиць Excel

xlwt (http://pypi.org/project/xlwt) - бібліотека для створення електронних таблиць у форматі Microsoft Excel 95-2003 на будь-якій платформі. В прикладі за допомогою xlwt 1.3.0 створюється робоча книга і лист Excel, в комірки якого заноситься значення і формула.

```
import xlwt
workbook = xlwt.Workbook() # робоча книга Excel
sheet = workbook.add_sheet("Sheet1") # робочий лист
sheet.write(0, 0, 7.0) # записати в комірку 0, 0
значення
sheet.write(0, 1, xlwt.Formula("A1+2")) # записати в
комірку 0, 1 формулу
workbook.save("Book1.xls") # зберегти файл
```

pywin32 - інтерфейс до win32 GUI API

Python for Win32 Extensions (pywin32) -це бібліотека, яка забезпечує доступ до багатьох Windows API з мови Python (http://github.com/mhammond/pywin32). Після установки документація доступна у файлі PyWin32.chm.

Приклад показує можливість застосування pywin32 (версія 221) для управління графічним інтерфейсом інших програм. Програма створює процес calc.exe, знаходить вікно програми і імітує натискання клавіш клавіатури і миші користувачем. Після цього програма входить в цикл, в якому показує відносні координати миші. Щоб завершити програму посуньте курсор миші в верхній лівий кут екрану.

```
import os, sys, time, win32api, win32gui, win32con
os.system('start calc.exe') # виконати команду і
продовжити роботу
time.sleep(1) # чекати 1 секунду
hwnd = win32gui.FindWindow(None, u"Калькулятор") #
знайти дескриптор вікна за назвою
try:
    win32gui.SetForegroundWindow(hwnd) # установити
на передній план
    cw=win32gui.GetWindowRect(hwnd) # координати
вікна
except:
    sys.exit()
for k in [0x32,0x6B,0x33,0x0D]: # натиснути клавіші 2
+ 3 Enter
    win32api.keybd_event(k,0,0,0) # k - віртуальний
код клавіші
    time.sleep(0.1)

win32api.SetCursorPos([cw[0]+380,cw[1]+260]) #
установити курсор миші
time.sleep(1)
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,0,
0,0,0) # натиснути ліву клавішу миші
time.sleep(1)
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,0,0,
0,0) # відпустити ліву клавішу миші
```

```
time.sleep(1)

while True: # цикл
    c=win32gui.GetCursorInfo() # координати курсора миші
    print c[2][0]-cw[0], c[2][1]-cw[1] # відносні координати
    if c[2]==(0,0): break # завершити якщо координати (0,0)
```

```
305 68
295 71
```

win32com.client - об’єкти Excel

COM (Component Object Model) — платформа компонентно-орієнтованого програмування, яка використовується в ОС Windows. Підтримує повторене використання і можливість взаємодії об’єктів незалежно від мови програмування, на якій вони були розроблені. Основними елементами COM є: об’єкт COM (екземпляр класу COM в сервері COM), сервер COM (програма, яка організовує доступ до створеного в ній об’єкта COM, реалізуючи інтерфейси), клієнт COM (програма, яка, використовуючи інтерфейс, отримує доступ до об’єкта COM), інтерфейс COM (визначає відкриті методи, які використовуються для доступу до об’єкта COM), клас COM (реалізація інтерфейсу COM в сервері COM). Пакет `win32com` є частиною бібліотеки pywin32 і реалізує пітримку COM в Python. В прикладі створюється клієнт COM для доступу до об’єктів Excel.

```
import win32com.client # імпортувати модуль win32com.client
obj = win32com.client.Dispatch("Excel.Application") # створити об'єкт Excel.Application
obj.Visible = 1 # зробити Excel видимим
obj.Workbooks.Add() # додати робочу книгу
```

```
obj.Cells(1,1).Value = "Hello" # в комірку 1,1
помістити "Hello"
```

	A	B	C
1	Hello		
2			

Рисунок 53 - Результат роботи програми в Excel

win32com.client - об'єкти Excel з обробкою подій

В більш складному прикладі створюється клієнт COM для доступу до об'єктів Excel з обробкою подій. Під час оброблення події програми `OnSheetBeforeDoubleClick` та події робочого листа `OnSelectionChange` виводиться інформація про вибрані комірки. Цей модуль слід виконувати так:

```
python main.py
```

Для виходу слід в консолі натиснути Esc. Дивись інші приклади в `c:\Python27\Lib\site-packages\win32com\test`.

```
import win32com.client
import msvcrt, pythoncom

class MyExcelEvents: # події прикладної програми
Excel
    def OnSheetBeforeDoubleClick(self, Sheet, Target,
Cancel): # обробник події OnSheetBeforeDoubleClick
        print "SheetBeforeDoubleClick"
        print Target.GetAddress() # Target - комірка
        print Target.Column # колонка
        print Target.Row # рядок
        #Target.Value='111' # значення комірки

class MyWorksheetEvents(): # події робочого листа
Worksheet
```

```
    def OnSelectionChange(self,Range): # обробник
подій OnSelectionChange
        print dir(Range)[:3] # вивести деякі атрибути
об'єкта Range (діапазон комірок)
        print Range.GetAddress() # отримати адресу
комірки
        print Range.GetValue() # отримати значення
комірки

excelApp =
win32com.client.DispatchWithEvents("Excel.Application
",  MyExcelEvents) # створити об'єкт
Excel.Application з обробкою подій
#excelApp =
win32com.client.Dispatch("Excel.Application") # без
обробки подій

excelApp.Visible = 1 # зробити Excel видимим
workBook=excelApp.Workbooks.Add() # додати робочу
книгу

workSheet=excelApp.ActiveWorkbook.ActiveSheet #
активний лист
workSheet=win32com.client.DispatchWithEvents(workShee
t, MyWorksheetEvents) # створити об'єкт workSheet з
обробкою подій

workSheet.Cells(1,1).Value = 100 # в комірку 1,1
помістити 100
# в діапазони комірок помістити значення:
workSheet.Range("B1:D2").Value =((1,2,3),(10,20,30))
workSheet.Range("B3:D3").Value =(u"а",u"б",u"в")
workSheet.Range("A2").Value = "=A1+1"
# або workSheet.Cells(2,1).Formula = "=A1+1"
```

```
while True: # цикл
    if msvcrt.kbhit(): # якщо в консолі натиснута клавіша
        if ord(msvcrt.getch())==27: break # завершити, якщо це Esc
    pythoncom.PumpWaitingMessages() # обробляти події

workBook.Close(False) # закрити робочу книгу без збереження
excelApp.Quit() # вийти з Excel

excelApp=workBook=workSheet=None
from win32com.test.util import CheckClean
CheckClean() # перевірити скільки COM об'єктів залишилося
pythoncom.CoUninitialize() # відмінити ініціалізацю
CheckClean() # перевірити скільки COM об'єктів залишилося
```

```
SheetBeforeDoubleClick
$A$1
1
1
['Activate', 'AddComment', 'AdvancedFilter']
$A$2
101.0
```

A2 fx =A1+1

	A	B	C	D
1	100	1	2	3
2	101	10	20	30
3		а	б	в

Рисунок 54 - Робочий лист Excel

win32com.client - об'єкти SOLIDWORKS

В прикладі створюється клієнт COM для доступу до об’єктів системи автоматизованого проектування SOLIDWORKS. Програма змінює значення розміру активної моделі і перебудовує модель.

```
import win32com.client
swApp =
win32com.client.Dispatch("SldWorks.Application") #
створити об'єкт SldWorks.Application
Part=swApp.ActiveDoc # активний документ
# змінити значення параметра "D1@Extrude1" на 10 мм
Part.Parameter("D1@Extrude1").SystemValue = 10.0/1000
Part.EditRebuild # перебудувати модель
```

pyserial - доступ до послідовного порту

pyserial (http://github.com/pyserial/pyserial) - модуль Python для доступу до послідовного порту на Windows, OSX, Linux, BSD (будь-які POSIX системи) і IronPython. Для тестування віртуальних послідовних портів можна використовувати Eltima Virtual Serial Port Driver (http://www.eltima.com/ru/products/vspdxp) або com0com (http://com0com.sourceforge.net) і створити з’єднані віртуальні порти COM6-COM7. За допомогою Serial Port Terminal можна відкрити COM6 або COM7 для запису чи читання даних. Або можна записувати і читати за допомогою pyserial 2.7, як це показано в прикладі.

```
import serial,time
data=list("hello!") # дані, що будуть надсилатись
ser6 = serial.Serial(port='COM6', baudrate=9600) #
відкрити порт COM6
print ser6.portstr # перевірити чи порт
використовується
ser7 = serial.Serial(port='COM7', baudrate=9600) #
```

```
відкрити порт COM7
print ser7.portstr
x=''
while x!='!': # поки на COM6 не прийде байт '!'
    x=data.pop(0) # отримати і видалити перший
елемент
    ser7.write(x) # послати дані з COM7 на COM6
    time.sleep(1) # чекати 1 секунду
    x=ser6.read(1) # читати байт на COM6
    print x,
ser7.close() # закрити порт
ser6.close() # закрити порт
```

```
COM6
COM7
h e l l o !
```

pyFirmata - комунікація комп'ютера та Arduino

Arduino (www.arduino.cc) - відкрита і зручна у використанні платформа, яка основана на одноплатному мікроконтроллері Atmel AVR і використовується аматорами для побудови простих систем автоматики і робототехніки [13]. Firmata (http://firmata.org) це загальний протокол для зв'язку мікроконтроллерів з головним комп'ютером. Firmata дозволяє експериментувати з Arduino без необхідності його перепрограмовування кожного разу. В прикладі використано плату Arduino UNO для вимірювання значень температури за допомогою терморезистора і виведення їх на графік у реальному часі.

Передусім установіть драйвер USB-SERIAL для Arduino. В цьому прикладі це CH340 (http://www.wch.cn/download/CH341SER_ZIP.html). Розпакуйте на комп'ютер середовище Arduino IDE. У файлі /avr/boards.txt перевірте швидкість передачі даних `uno.upload.speed=57600`. Під'єднайте датчик температури (терморезистор) до контактів GND, ANALOG IN 0 (A0), 5V (рис.). Під'єднайте світлодіод до

контактів GND і DIGITAL 13. Під’єднайте Arduino до USB-порту комп’ютера. В гілці “порти” диспетчера пристроїв знайдіть USB-SERIAL CH340 (COM9), де COM9 - назва послідовного порту. У вас номер може бути інший. З Arduino IDE завантажте в пам’ять мікроконтроллера приклад Firmata/StandardFirmata. Установіть на комп’ютері pyFirmata (https://github.com/tino/pyFirmata) і запустіть наступний приклад.

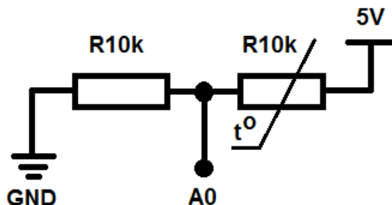


Рисунок 55 - Під’єднання датчика температури

```
import matplotlib.pyplot as plt
import time
from pyfirmata import Arduino, util
board = Arduino('COM9') # з'єднати Arduino з портом COM9
it = util.Iterator(board); it.start() # для використання аналогових портів
board.analog[0].enable_reporting()
X=[] # список зі значеннями температури
plt.ion() # інтерактивна побудова графіка
while len(X)<30: # поки довжина списку мала
    time.sleep(1) # затримка 1 с
    x=board.analog[0].read() # читати значення з аналогового входу 0
    print x
    X.append(x) # додати в список
    plt.plot(X,'ko-'); plt.draw() # рисувати графік (plt.clf() - очистити)
    if x>0.35: # якщо температура висока
```

```
        board.digital[13].write(1) # включити
світлодіод
    else:
        board.digital[13].write(0) # виключити
світлодіод
board.exit() # вийти
```

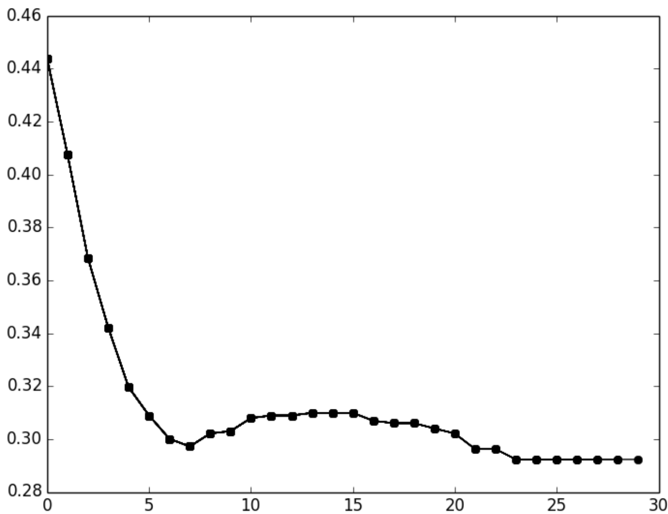


Рисунок 56 - Графік температури

concurrent.futures - запуск паралельних задач

concurrent.futures - високорівневий інтерфейс для асинхронного виконання виконуваних об'єктів за допомогою потоків (`ThreadPoolExecutor`) або процесів (`ProcessPoolExecutor`). Входить в стандартну бібліотеку Python 3.2, але доступний і для Python 2.7 (http://pypi.org/project/futures). Подібний приклад можна також розробити за допомогою multiprocessing. Див. також приклад

використання інтерфейсу futures в розподілених обчисленнях (Dask.Distributed).

```
import concurrent.futures, time
def f(x): # функція, яка буде виконуватись в окремих
процесах
    time.sleep(x) # затримка (тільки для тестування
паралельності)
    return x
if __name__ == '__main__':
    with
concurrent.futures.ProcessPoolExecutor(max_workers=4)
as e:
        a=e.submit(f, 4) # виконати в окремому
процесі f(x=4)
        b=e.submit(f, 2)
        c=e.submit(f, 3)
        d=e.submit(f, 1)
        # для кожного об'єкта Future, що виконує f
        for fut in
concurrent.futures.as_completed([a,b,c,d]):
            print fut.result(), # отримати результати
асинхронно
        #print [x.result() for x in [a,b,c,d]] # або
чекати усі результати
        #print [x for x in e.map(f, [1,2,3,4])] # або
простіше
```

```
1 2 3 4
```

Dask - розподілені обчислення на чистій Python

Dask 0.18.2 (http://dask.pydata.org) - це гнучка бібліотека для паралельних обчислень. Dask забезпечує динамічне планування задач, оптимізоване для інтерактивних обчислювальних навантажень, та прості шляхи масштабування задач Pandas, Scikit-

Learn і Numpy. Дозволяє легко паралелізувати довільний Python-алгоритм шляхом створення “лінивих” функцій з відкладеним виконанням. Функція `dask.delayed` обгортає довільну функцію так, що вона не виконується миттєво, а створює граф задач. Передача відкладених результатів іншим відкладеним функціям створює залежності між задачами. Обчислити результати паралельно можна за допомогою методу `compute`. Нижче наведено приклад алгоритму, який паралелізується. Для виконання прикладу на кластері див. приклад Dask.Distributed.

```
from dask.distributed import Client
from dask import delayed
import time
def f(x): # функція, яка буде виконуватись в окремих процесах
    time.sleep(x)
    return x
if __name__ == '__main__':
    #client = Client() # клієнт (кластер на локальній машині)
    client = Client('192.168.1.33:8786') # клієнт
    res=[]
    for x in [1,2,3,4]:
        f_ = delayed(f)(x) # відкладене виконання функції f
        res.append(f_) # додати відкладений результат в список
    sum_=delayed(sum)(res) # відкладене виконання функції sum
    print sum_.compute() # обчислити f(1),f(2),f(3),f(4) паралельно, а потім обчислити їх суму
```

```
10
```

Dask.Distributed - розподілені обчислення

Розподілені обчислення - це вид паралельних обчислень за допомогою множини комп'ютерів, які об'єднані в мережу. Dask.Distributed (http://distributed.readthedocs.io) - це легка бібліотека для розподілених обчислень на Python. Вона розширює API `concurrent.futures` і `Dask` (бібліотека паралельних обчислень на чистій Python) для невеликих кластерів. Для виконання прикладу необхідно установити Dask повністю на кожній Windows машині:

```
pip install "dask[complete]"
```

Або на Linux-машині:

```
sudo pip2 install "dask[complete]"
```

На одній машині (наприклад 192.168.1.33) запустити планувальник:

```
dask-scheduler
```

На кожній машині запустити виконавців, які виконують завдання планувальника за допомогою ThreadPool. Якщо обчислення вивільняють GIL (наприклад NumPy або Pandas), введіть:

```
dask-worker 192.168.1.33:8786
```

Або, якщо обчислення не вивільняють GIL:

```
dask-worker 192.168.1.33:8786 --nprocs 4 --nthreads 1
```

Виконати програму клієнта:

```
python main.py
```

Переглянути статус виконання можна в браузері (потрібен установлений Bokeh):

```
http://192.168.1.33:8787
```

```
from dask.distributed import Client, as_completed
import time
def f(x): # функція, яка буде виконуватись в окремих процесах
    time.sleep(x)
    return x
if __name__ == '__main__':
    #client = Client() # клієнт (кластер на локальній машині)
    client = Client('192.168.1.33:8786') # клієнт
    print client
    a=client.submit(f, 4) # виконати на кластері f(x=4)
    b=client.submit(f, 2)
    c=client.submit(f, 3)
    d=client.submit(f, 1)
    # для кожного об'єкта Future, що виконує f
    for fut in as_completed([a,b,c,d]):
        print fut.result(), # отримати результати асинхронно
    #futures=client.map(f, [1,2,3,4,5,6,7,8]) # або
    #print client.gather(futures)  # чекати усі результати
    # або [fut.result() for fut in futures]
```

```
<Client: scheduler='tcp://192.168.1.33:8786' processes=8 cores=8>
1, 2, 3, 4
```

PyQt4 - фреймворк Qt в Python

PyQt4 (http://www.riverbankcomputing.com /software /pyqt) - це “прив’язка” фреймворку Qt до мови Python. Qt - це багатоплатформовий програмний фреймворк для створення ПЗ

мовою C++. Містить класи для створення GUI, роботи з мережею, базами даних, OpenGL, мультимедіа і т.д. Існує також прив’язка з більш вільними умовами ліцензування PySide (http://wiki.qt.io/PySide), яка сумісна на рівні API з PyQt. В прикладі показано програму з GUI для розрахунку квадрату числа. Цей приклад буте також працювати в PySide 1.2.4, якщо замінити рядок `PyQt4` на `PySide`.

```
import sys
from PyQt4.QtCore import * # базові класи
from PyQt4.QtGui import * # GUI класи
class My_Dialog(QDialog): # клас вікна успадковує
QDialog
    def __init__(self, parent=None): # конструктор
        super(My_Dialog, self).__init__(parent) #
виклик конструктора QDialog
        self.setWindowTitle("x**2") # надпис вікна
        self.resize(150, 100) # змінити розмір вікна
        self.pushButton =
QPushButton("Calculate",self) # кнопка
        self.pushButton.setGeometry(QRect(25, 50, 90,
30)) # змінити геометрію кнопки
        self.lineEdit = QLineEdit("2",self) # поле
редагування
        self.lineEdit.setGeometry(QRect(25, 10, 90,
30)) # змінити геометрію поля редагування
        self.lineEdit.setFocus() # установити фокус
вводу
        # приєднати сигнал clicked() до слота
self.slot
        self.connect(self.pushButton,
SIGNAL("clicked()"), self.slot)
    def slot(self): # обробник сигналу clicked()
        x=float(self.lineEdit.text()) # введене у
поле число
```

```
        self.lineEdit.setText((x**2).__str__()) # вивести у поле
app = QApplication(sys.argv) # створити застосування
dialog = My_Dialog() # створити вікно
dialog.show() # показати вікно
app.exec_() # виконати застосування
```

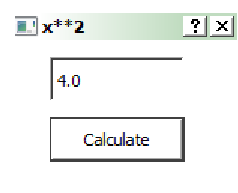


Рисунок 57 - Вікно програми

PyQt4 - елементи керування QtGui

Більш складний приклад використання таких елементів керування як `QMainWindow` (головне вікно), `QMenuBar` (смуга меню), `QMenu` (меню), `QAction` (дія GUI), `QTextBrowser` (текстовий браузер), `QComboBox` (список), `QDial` (пристрій регулювання), `QCheckBox` (прапорець), `QPixmap` (рисунок), `QLabel` (надпис або рисунок), `QTreeWidget` (дерево), `QTreeWidgetItem` (елемент дерева), `QFileDialog` (вікно вибору файлу), `QMessageBox` (вікно з повідомленням), `QApplication` (GUI-застосування). Програма Qt Designer дозволяє полегшити створення складних GUI в режимі WYSIWYG. Після створення нею файлу опису GUI `main.ui` потрібно згенерувати код Python за допомогою програми `pyuic`:

```
pyuic.py -x main.ui -o main.py
```

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *
```

```
class MyWindow(QMainWindow): # клас вікна успадковує
QMainWindow
    def __init__(self, parent=None): # конструктор
        super(MyWindow, self).__init__(parent) #
виклик конструктора QMainWindow
        self.resize(400, 300) # змінити розмір вікна
        self.menubar = QMenuBar(self) # створити
смугу меню
        self.menubar.setGeometry(QRect(0, 0, 400,
24)) # геометрія
        # підменю:
        self.menuFile = QMenu(self.menubar) # меню
File в menubar
        self.menuFile.setTitle("File") # установити
надпис
        self.menuNew = QMenu(self.menuFile) # меню
New в menuFile
        self.menuNew.setTitle("New")
        self.menuAbout = QMenu(self.menubar)
        self.menuAbout.setTitle("About")
        # дії меню:
        self.actionNewItem = QAction(self)
        self.actionNewItem.setText("New Item")
        self.actionOpen = QAction(self)
        self.actionOpen.setText("Open")
        # додати до меню дії:
        self.menuNew.addAction(self.actionNewItem)

self.menuFile.addAction(self.menuNew.menuAction())
        self.menuFile.addAction(self.actionOpen)

self.menubar.addAction(self.menuFile.menuAction())

self.menubar.addAction(self.menuAbout.menuAction())
```

```
        self.textBrowser = QTextBrowser(self) # 
текстовий браузер
        self.textBrowser.setGeometry(QRect(10, 30, 
130, 200))
        self.comboBox = QComboBox(self) # 
комбінований список
        self.comboBox.addItems([str(i) for i in 
range(1,11)]) # додати елементи
        self.comboBox.setGeometry(QRect(10, 240, 100, 
20))
        self.dial = QDial(self) # пристрій 
регулювання
        self.dial.setNotchesVisible(True) # 
установити шкалу
        self.dial.setGeometry(QRect(150, 30, 50, 50))
        self.checkBox=QCheckBox("CheckBox",self) # 
прапорець
        self.checkBox.setTristate(True) # установити 
три можливі стани
        self.checkBox.setGeometry(QRect(150, 110, 
100, 20))
        self.pixmap=QPixmap() # рисунок
        self.pixmap.load(u"pic.png") # завантажити
        self.label=QLabel(self) # надпис
        self.label.setPixmap(self.pixmap) # 
установити рисунок
        self.label.setGeometry(QRect(150, 150, 100, 
100))
        self.treeWidget = QTreeWidget(self) # дерево
        self.treeWidget.setGeometry(QRect(250, 30, 
140, 200))
        item_0 = QTreeWidgetItem(self.treeWidget) # 
додати кореневий елемент
        item_1 = QTreeWidgetItem(item_0) # додати 
дочірній елемент до item_0
```

```
        item_0.setText(0,'editable') # установити
текст елемента
        item_1.setText(0,'11')
        item_0.setFlags(Qt.ItemIsSelectable |
Qt.ItemIsEditable | Qt.ItemIsDragEnabled |
Qt.ItemIsUserCheckable | Qt.ItemIsEnabled) #
властивості елемента
        self.treeWidget.expandToDepth(2) # розвернути
дерево до рівня 2
        # приєднати сигнали до слотів:

self.connect(self.comboBox,SIGNAL("currentIndexChange
d(int)"), self.slot2)
        self.connect(self.dial,
SIGNAL("valueChanged(int)"), self.slot3)
        self.connect(self.checkBox,
SIGNAL("stateChanged(int)"), self.slot4)
        self.connect(self.actionOpen,
SIGNAL("triggered()"), self.slot1)
        self.connect(self.actionNewItem,
SIGNAL("triggered()"), self.slot2)
        self.connect(self.treeWidget,
SIGNAL("itemDoubleClicked(QTreeWidgetItem*,int)"),
self.slot4)

    #обробники відповідних сигналів:
    def slot1(self):
        filename=QFileDialog.getOpenFileName(self,
"MyFile") # ім'я файлу з вікна вибору файлу
        self.textBrowser.append("<font color=blue>" +
filename + "</font>") # додати в текстовий браузер

    def slot2(self):
        if self.treeWidget.currentItem()!=None: #
якщо існує поточний елемент дерева
```

```
            item =
QTreeWidgetItem(self.treeWidget.currentItem()) #
створити дочірній елемент до поточного
            item.setText(0,'New') # його текст
            item.setTextColor(0,QColor(255,0,0)) #
його колір
            item.setCheckState(0, Qt.Checked) #
установити стан
        else: # інакше вивести вікно повідомлення
            QMessageBox.warning(self,
"MessageBox","select parent item!")
        self.textBrowser.append("index {0} - text
{1}".format(self.comboBox.currentIndex(),
self.comboBox.currentText())) # додати в браузер
індекс і текст поточного елемента списку

    def slot3(self):

self.checkBox.setText(self.dial.value().__str__()) #
значення пристрою регулювання

    def slot4(self):
        if self.treeWidget.currentItem()!=None: #
якщо існує поточний елемент дерева
            if
self.treeWidget.currentItem().checkState(0): # якщо
стан вибраний

self.treeWidget.removeItemWidget(self.treeWidget.curr
entItem(),0) # видалити поточний елемент
        self.textBrowser.append("checkBox
{0}".format(self.checkBox.checkState())) # додати в
браузер стан перемикача

app = QApplication(sys.argv) # створити застосування
```

```
label = QLabel("<font color=red size=72><b>" +
"Hello" + "</b></font>") # надпис (з'являється перед
появою головного вікна)
label.setWindowFlags(Qt.SplashScreen) # властивості
вікна
label.show() # показати
import time
time.sleep(1) # затримка на 1 с.
label.hide() # сховати надпис
window = MyWindow() # створити вікно
window.show() # показати вікно
#QTimer.singleShot(10000, app.quit) # вийти через 10
с.
app.exec_() # виконати застосування
```

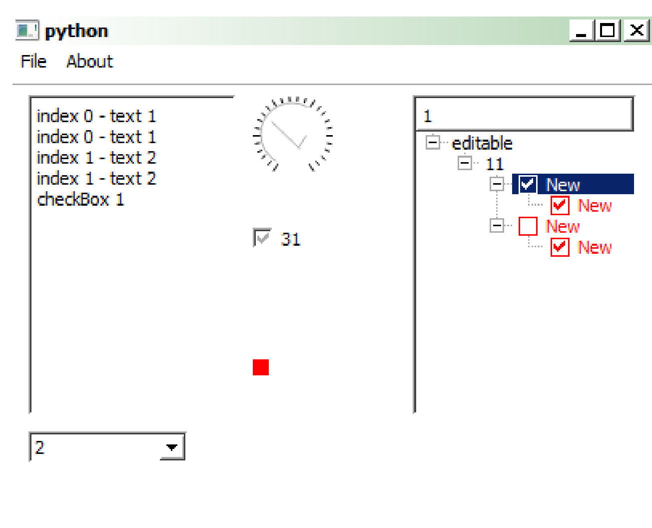


Рисунок 58 – Вікно програми

PyQt4 - створення елемента керування

В прикладі показано створення нового елемента керування (GUI-віджету) `MyButton` шляхом успадкування класу `QPushButton`

(кнопка). На відміну від базового класу нова кнопка володіє атрибутом `state`, логічне значення якого змінюється на протилежне під час натиску на неї. Крім того це значення відображається на самій кнопці.

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

class MyButton(QPushButton): # клас успадковує
QPushButton
    state = True
    def __init__(self, state,parent=None): #
конструктор
        super(MyButton, self).__init__(parent) #
виклик конструктора QPushButton
        self.state=state # стан кнопки (True, False)
        self.setText(self.state.__str__()) #
установити надпис на кнопці
        # приєднати сигнал clicked() до слота
self.change_state()
        self.connect(self, SIGNAL("clicked()"),
self.change_state)
    def change_state(self): # обробник сигналу
clicked()
        if self.state: # якщо стан True
            self.emit(SIGNAL("state_true"),
self.state) # генерувати сигнал state_true
            self.state=False # змінити стан
        else: # інакше генерувати сигнал state_false
            self.emit(SIGNAL("state_false"),
self.state)
            self.state=True # змінити стан
        self.setText(self.state.__str__()) #
установити надпис на кнопці
```

```
class My_Dialog(QDialog): # клас вікна успадковує
QDialog
    def __init__(self, parent=None): # конструктор
        super(My_Dialog, self).__init__(parent) #
виклик конструктора QDialog
        self.resize(230, 100) # змінити розмір вікна
        self.pushButton1 = MyButton(True,self) #
кнопка
        self.pushButton1.setGeometry(QRect(25, 50,
90, 30)) # змінити геометрію кнопки
        self.pushButton2 = MyButton(False,self) #
кнопка
        self.pushButton2.setGeometry(QRect(120, 50,
90, 30)) # змінити геометрію кнопки
        self.lineEdit = QLineEdit(self) # поле
редагування
        self.lineEdit.setGeometry(QRect(25, 10, 90,
30)) # змінити геометрію поля редагування
        # приєднати сигнали до слотів
        self.connect(self.lineEdit,
SIGNAL("textChanged(QString)"),
                     self,
SLOT("setWindowTitle(QString)"))
        self.connect(self.pushButton1,
SIGNAL("state_true"), self.slot)
        self.connect(self.pushButton2,
SIGNAL("state_true"), self.slot)
    def slot(self): # обробник сигналу state_true
        button = self.sender() # компонент, що
надіслав сигнал
        # якщо це ніякий компонент або не об'єкт
класу MyButton
        if button is None or not isinstance(button,
MyButton):
```

```
            return # то вийти
        global x # звернення до глобальної змінної
        if button==self.pushButton1: x+=1 # якщо
кнопка pushButton1
        else: x-=1 # інакше
        self.lineEdit.setText(x.__str__())
x=0 # глобальна змінна
app = QApplication(sys.argv) # створити застосування
dialog = My_Dialog() # створити вікно
dialog.show() # показати вікно
app.exec_() # виконати застосування
```

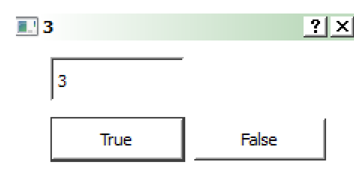


Рисунок 59 - Вікно програми

PyParsing - зручний синтаксичний аналіз

Синтаксичний аналіз (парсинг) - це процес зіставлення послідовності лексем (неподільних груп символів) певної мови з її формальною граматикою (способом опису мови). Лексеми отримуються шляхом лексичного аналізу (токенізації). PyPasing 2.2.0 (http://pypi.org/project/pyparsing) - модуль для синтаксичного аналізу, який реалізує альтернативний і більш зручний підхід для створення і використання граматик, у порівнянні з традиційним lex/yacc або використанням регулярних виразів. Модуль містить класи для створення граматик прямо в Python-коді.

```
from __future__ import print_function
from pyparsing import *
s="hello world!" # текст, який будемо парсити
# вираз для парсингу:
```

```
word=Word(alphas) # слово з букв,
alphas='abcde...'+'ABCDE...'
print(type(word))
#<class 'pyparsing.Word'>
p="hello"+word+Literal("!")
# або p=And([Literal("hello"), word, Literal("!")])
print(type(p))
#<class 'pyparsing.And'>
try: # тут бажане перехоплення помилки
    print(p.parseString(s)) # знайти p в s ОДИН раз
#['hello', 'world', '!']
    print(p.parseString(s).asList()[0])
#hello
except: pass

for x in Word(alphas).searchString(s): # знайти УСІ
слова в s
    print(x)
#['hello']
#['world']

print(ZeroOrMore("o").searchString(s)) # 0 або більше
#[['o'], [], ['o']]
print(OneOrMore("o").searchString(s)) # 1 або більше
#[['o'], ['o']]
res=[x for x in OneOrMore("o").scanString(s)]
print(res[0]) # кортеж
#((['o'], {}), 4, 5)

p=Literal('hello')^Literal('world') # або 'hello' або
'world'
# або p=Or([Literal('hello'), Literal('world')])
print(p.parseString("world"))
#['world']
```

```
p=Literal("world")|Literal("hello") # знайти перше
world або hello
# або MatchFirst([Literal("world"),Literal("hello")])
print(p.parseString(s))
#['hello']

p=Literal("world")&Literal("hello") # як +, але не
послідовно, а в довільному порядку
# або p=Each([Literal("world"), Literal("hello")])
print(p.parseString(s))
#['hello', 'world']

# літерал і слово (якщо є; але без нього)
p=Literal('hello')+Optional(Suppress(Word(alphas)))
print(p.parseString(s)) # без другого слова
#['hello']

def fn(s, loc, toks):
    print(s, loc, toks)
p=Word(alphas).setParseAction(fn)+Word(alphas) #
викликає функцію fn для першого слова
p.parseString(s)
#hello world! 0 ['hello']

p=Word(alphas).setResultsName("word1")+Word(alphas+'!
') # у виразі задано ім'я для першого слова
print(p.parseString(s).word1)
#hello

p=Literal("hello").setParseAction(replaceWith("hi"))
print(p.transformString(s)) # заміна hello на hi
#hi world!

alphasUA='АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯабвгґдеєжз
иіїйклмнопрстуфхцчшщьюя'
```

```
alphasRU='АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзи
йклмнопрстуфхцчшщъыьэюя'
print(Word(alphasUA).parseString("привіт world!")[0])
#привіт

# іменовані регулярні вирази
print(Regex(r"hello
(?P<name>.*)").parseString(s).name)
#world!

print(SkipTo(Literal("world")).parseString(s)) # все
що до літералу "world"
#['hello ']

p=Combine(Word(alphas)+Literal(" ")+Word(alphas)) #
об'єднати
print(p.parseString(s))
#['hello world']

w = Word(nums)
p = Forward() # попередня декларація
p << ('['+OneOrMore(Group(p)^w)+']') # рекурсія
print(p.parseString('[1 2 [3 4]]'))
#['[', '1', '2', ['[', '3', '4', ']'], ']']
```

pymorphy2 - морфологічний аналізатор

pymorphy2 (https://github.com/kmike/pymorphy2) - морфологічний аналізатор для російської і української мови. Повертає граматичну інформацію про слово (число, рід, відмінок, частина мови і т.д.), приводить слово до нормальної форми, ставить слово в потрібну форму. Використовує словник opencorpora.org. Для незнайомих слів будуються гіпотези. Дивись також NLTK (www.nltk.org) - пакет для обробки природної мови. Ці пакети широко застосувуються в галузі штучного інтелекту.

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer() # об'єкт для
морфологічного аналізу
pp=morph.parse(u'уменьшить') # виконати аналіз слова
p=pp[0] # перший варіант слова
t=p.tag # набір грамем слова
print t.POS # частина мови
#див. також t.tense (час), t.case (відмінок),
t.gender (рід), t.number (число), t.mood (спосіб
дієслова), t.стан (voice) та ін.
n=p.normalized # нормальна форма
print n.word
print n.inflect({'VERB','3per','futr'}).word #
перевести слово в іншу форму
for x in p.lexeme: # лексема (різні форми слова)
    print x.word,
```

```
INFN
уменьшить
уменьшит
уменьшить уменьшил уменьшила уменьшило ...
```

pygments - підсвітка синтаксису

Підсвітка синтаксису - це виділення синтаксичних конструкцій тексту за допомогою різних шрифтів, їх кольорів і написань. Використовується для спрощення сприйняття тексту. Підсвітка синтаксису виконується за допомогою лексичного аналізатора, який визначає окремі лексеми (послідовність символів, що має певне значення). Пакет `pygments` 2.2.0 (http://pygments.org) призначений для підсвічування синтаксису, підтримує близько 300 мов, дозволяє створювати нові лексичні аналізатори, виводить в багатьох форматах (HTML, RTF, LaTeX та ін.), може використовуватись в командному рядку або як бібліотека.

```
from pygments import highlight # повертає
відформатований текст
from pygments.lexers import PythonLexer # лексичний
аналізатор Python
from pygments.formatters import HtmlFormatter # для
форматування у вигляді HTML
code = u'print "Hello World" # коментар' # Юнікод (!)
рядок Python коду
print highlight(code, PythonLexer(), HtmlFormatter())
# повертає проаналізований  PythonLexer() та
відформатований HtmlFormatter() текст
#print HtmlFormatter().get_style_defs('.highlight') #
повертає текст стилю CSS

# все в одному файлі HTML
#print highlight(code, PythonLexer(),
HtmlFormatter(full=True))

# все в одному файлі HTML окрім стилю. Стиль окремим
файлом python.css
#print highlight(code, PythonLexer(),
HtmlFormatter(full=True,cssfile='python.css'))

# новий лексичний аналізатор на основі регулярних
виразів
from pygments.lexer import RegexLexer
from pygments.token import *
class MyLexer(RegexLexer):
    tokens = {'root': [(r'[^#]+', Text),(r'#.*\n',
Comment),(r'\n.*', Text)]} # послідовність (рег.
вираз, токен)
print highlight(code, MyLexer(), HtmlFormatter())
```

```
<div class="highlight"><pre><span></span><span
class="k">print</span> <span class="s2">&quot;Hello
World&quot;</span> <span class="c1"># коментар</span>
</pre></div>

<div class="highlight"><pre><span></span>print
&quot;Hello World&quot; <span class="c">#
коментар</span>
</pre></div>
```

pygments - підсвітка синтаксису в Tkinter

Приклад показує способи підсвітки синтаксису в текстовому віджеті `Tkinter.Text` за допомогою пакету `pygments`.

```
from Tkinter import *
from pygments.lexers import PythonLexer # лексичний
аналізатор Python
from pygments.formatters import RawTokenFormatter
# RawTokenFormatter - для форматування у "сирому"
вигляді: тип токена<TAB>repr(рядок токена)\n

root = Tk() # головне вікно
text = Text(root, font=('arial', 10, 'normal')) #
віджет для відображення тексту
text.pack() # розташувати
code = u'print "hello" # коментар' # рядок Python
коду
text.insert("end", code) # вставити текст в текстовий
віджет

# конфігурувати теги текстового віджету
text.tag_configure("Token.Keyword",
foreground='blue', font=('arial', 10, 'bold'))
text.tag_configure("Token.Text", foreground='black',
font=('arial', 10, 'normal'))
```

```
text.tag_configure("Token.Literal.String",
foreground='red', font=('arial', 10, 'normal'))
text.tag_configure("Token.Comment",
foreground='darkgreen', font=('arial', 10, 'normal'))

code = text.get("1.0", "end-1c") # отримати текст з
текстового віджету
text.delete("1.0", "end") # видалити весь текст з
текстового віджету

# перший спосіб:
from pygments import highlight # повертає
відформатований текст
for line in highlight(code, PythonLexer(),
RawTokenFormatter()).split("\n"): # для кожного рядка
тексту, відформатованого за допомогою PythonLexer()
та RawTokenFormatter()
    pair=line.split("\t") # розділити рядок символом
табуляції
    if pair!=['']: # якщо пара не пуста
        (token, s) = pair
        print token, eval(s) # вивести на консоль
        text.insert("end", eval(s), token) # вставити
текст з тегом в віджет

# другий спосіб:
#from pygments import lex # лексичний аналізатор,
повертає ітератор токенів
#for token, content in lex(code, PythonLexer()):
#    print token, content
#    text.insert("end", content, str(token))

root.mainloop() # головний цикл програми
```

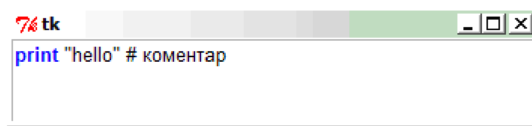


Рисунок 60 - Вікно програми

lxml - простий і швидкий парсинг XML і HTML

lxml 4.1.0 (http://lxml.de) - це Python-бібліотека для обробки XML і HTML, яка є прив'язкою до C бібліотек libxml2 і libxslt. Володіє повною підтримкою XML, є швидкою і зручною у використанні, сумісна з ElementTree API. В прикладі розглядається використання HTMLParser та xpath для отримання ключових слів некоректного HTML документу з тегу meta. XPath (XML Path Language) - це мова запитів до елементів XML документа.

```
from StringIO import StringIO
from lxml import etree
broken_html=r"""<html><head>
<meta content="Python, XML" name="keywords" />
<head>""" # некоректний документ HTML
parser=etree.HTMLParser()
tree=etree.parse(StringIO(broken_html), parser) #
парсинг
root=tree.getroot() # кореневий елемент (html)
es=root.findall('head/meta') # знайти усі теги meta
for e in es:
    if 'name' in e.attrib and 'content' in e.attrib:
# якщо є такі атрибути
        if e.attrib['name']=="keywords":
            print e.attrib['content']

# або за допомогою мови запитів xpath:
es=tree.xpath("head/meta[@name='keywords']/@content")
es=tree.xpath("child::head/child::meta[attribute::nam
```

```
e='keywords']/attribute::content") # або повний
синтаксис xpath
print es[0]
```

```
Python, XML
Python, XML
```

lxml - XSLT трансформації

XSLT (eXtensible Stylesheet Language Transformations) - це мова перетворення XML-документів. В прикладі за допомогою lxml до початкового документа XML застосовується таблиця стилів XSLT і отримується перетворений документ XML. Правила вибору даних з початкового документу створюються мовою запитів XPath.

```
from lxml import etree
from StringIO import StringIO
xslt_root = etree.XML('''\
 <xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:template match="/">
         <foo><xsl:value-of select="/a/b/text()"
/></foo>
     </xsl:template>
 </xsl:stylesheet>''') # таблиця стилів XSLT
transform = etree.XSLT(xslt_root) # функція
трансформації
f = StringIO('<a><b>Text</b></a>') # документ для
трансформації
doc = etree.parse(f) # парсинг документа
print transform(doc) # трансформований документ
```

```
<?xml version="1.0"?>
<foo>Text</foo>
```

Bottle - легкий WSGI веб-фреймворк

Bottle (http://bottlepy.org) - це швидкий, простий і легкий WSGI мікро веб-фреймворк для Python. Він розповсюджується як один файловий модуль і не має ніяких залежностей крім стандартної бібліотеки Python. Містить вбудований сервер і підтримує інші високопродуктивні WSGI сервери. В прикладі використано стару версію Bottle 0.5.8, яка працює навіть на PythonCE. Для тестування прикладу запустіть модуль і введіть в адресному рядку браузера один з URL, наведених нижче.

```
from bottle import route, run, request, send_file,
WSGIRefServer

# декоратор route() пов'язує функцію hello_world з
URL-адресами
@route('/') # http://localhost:8080/
@route('/index.html') #
http://localhost:8080/index.html
def hello_world():
    return '<h2>Hello World!</h2>' # повертає html
відповідь сервера

# тут :name означає будь-який текст.  Також можна
використовувати регулярні вирази
@route('/hello/:name') #
http://localhost:8080/hello/John
def hello_url(name):
    return 'Hello %s!' % name

# отримання GET параметра
@route('/hello') #
http://localhost:8080/hello?name=John
def hello_get():
    name = request.GET['name'] # отримати параметр
name
```

```
    return 'Hello %s!' % name

# html-форма
@route('/form') # http://localhost:8080/form
def form():
    return """<form action="/edit" method="post">
User: <input type="text" name="user">
Password: <input type="password" name="password">
<p>Text<Br><textarea name="text">John</textarea></p>
<input type="submit" value="Submit" /></form>"""

# відповідь після натиску кнопки submit на формі
@route('/edit', method='POST')
def hello_post():
    user = request.POST['user'] # значення поля user
    password = request.POST['password'] # значення
поля password
    users={'admin':'111'} # словник з парами
користувач:пароль
    if user in users and password==users[user]: #
якщо користувач і пароль коректні
        text = request.POST['text'] # значення поля
text
        return 'Hello %s!' % text
    else:
        return "Login failed" # помилка входу

# відсилання статичних файлів (html, jpg, png та ін.)
@route('/:filename#.*#') #
http://localhost:8080/static.html
http://localhost:8080/pic.png
def static_file(filename):
    send_file(filename, root='') # root - шлях до
статичних файлів
```

```
run(server=WSGIRefServer, host='localhost',
port=8080) # стартувати http сервер
```

```
127.0.0.1 - - [02/Aug/2018 14:28:48] "GET / HTTP/1.1"
200 21
127.0.0.1 - - [02/Aug/2018 14:29:06] "GET /index.html
HTTP/1.1" 200 21
127.0.0.1 - - [02/Aug/2018 14:29:20] "GET /hello/John
HTTP/1.1" 200 11
127.0.0.1 - - [02/Aug/2018 14:29:33] "GET /hello?name
=John HTTP/1.1" 200 11
127.0.0.1 - - [02/Aug/2018 14:30:12] "GET /form HTTP/
1.1" 200 222
127.0.0.1 - - [02/Aug/2018 14:30:42] "POST /edit HTTP
/1.1" 200 11
127.0.0.1 - - [02/Aug/2018 14:31:11] "GET /static.htm
l HTTP/1.1" 200 77
127.0.0.1 - - [02/Aug/2018 14:31:11] "GET /pic.png HT
TP/1.1" 200 75
```


Рисунок 61 - HTML-форма в браузері

РОЗДІЛ 3. ЗАДАЧІ

1. Створити програму для обчислення значення функції *f(x)*, якщо *i*=1 (рис.). Значення аргументу в програму передається з командного рядка.

$$f(x) = \frac{i + x}{ix + 2{,}5i} + \cos^{i+1}\left(x + \frac{\pi}{2}\right)$$

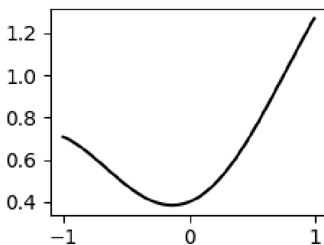


Рисунок 62 – Графік функції *f(x)*

2. Створити програму для обчислення значення функції *g(x),* якщо *a*=-0,5; *b*=0,5.

$$g(x) = \begin{cases} 0, & x < a \\ f(x), & a \le x \le b \\ x, & x > b \end{cases}$$

3. Розв'язати попередню задачу з використанням підпрограми-функції, яка повертає значення *g(x).*

4. Створити програму для виведення списку значень функції *g(x),* якщо *x* змінюється від -1 до 1 з кроком 0,1. Використати оператори циклу `for` або `while`.

5. Знайти наближене значення інтегралу $\int_{-1}^{1} f(x)dx$ з кроком інтегрування 0,1. Використати оператори циклу `for` або `while`.

6. Визначте тривалість обчислення значення функції *f(x)* і її інтегралу на вашому комп'ютері.

7. Знайти мінімальне і максимальне значення функції *f(x)* на проміжку [-1, 1], якщо крок *x* рівний 0,1. Використати оператори циклу і умови.

8. Знайти корінь рівняння *f(x)=1* на проміжку [-1,1], якщо крок *x* рівний 0,1. Використати оператори циклу і умови.

9. Розв’язати попередні задачі з використанням списків і функцій `sum`, `min`, `max`, `sorted`. Вивести списки, їх перше і останнє значення, довжини списків. Знайти значення аргумента *argmin* та *argmax*.
10. Створити словник, ключами якого є елементи кортежа (-1, 0, 1), а значеннями – відповідні значення функції *f(x)*. Вивести усі ключі і значення.
11. Створити довільний рядок. За допомогою операторів циклу і умови замінити задані символи (або слова) в рядку на інші. Розв’язати цю ж задачу за допомогою методу `replace`.
12. Створити довільний рядок. За допомогою операторів циклу і умови замінити текст в дужках на три крапки (`...`). Розв’язати цю ж задачу за допомогою модуля `re`.
13. Дано множини A={1,2,3,4} і B={3,4,5,6}. Знайти об’єднання, перетин, різницю і симетричну різницю цих множин.
14. Створити рекурсивну функцію *f(x,n)*, де *n* – додатне ціле число:

$$f(x,n)=\begin{cases} x^2, & n=0 \\ f(x^2, \quad n-1), & n\neq 0. \end{cases}$$

15. Створити генератор для утворення послідовності x^1, x^2, x^3, x^4.
16. Записати у текстовий файл значення аргументу *x* і функції *f(x)*, якщо *x* змінюється від -1 до 1 з кроком 0,1. Виконати пошук у створеному файлі значення *f(0)*.
17. Розв’язати попередню задачу з використанням модуля `csv`.
18. За допомогою модуля `pickle` записати у бінарний файл списки значень аргументу *x* і функції *f(x)*. Виконати пошук у створеному файлі значення *f(0)*.
19. Створити клас, який містить коструктор, атрибути-дані *xmin*, *xmax*, *dx* і атрибути-методи *f(x)*, *X()*, *F()*, які, відповідно, повертають значення функції, список значень аргумента *x*, список значень функції *f(x)*. Створити об’єкти класу.
20. Створити клас шляхом успадкування класу з попередньої задачі, який додатково містить методи, що повертають значення похідної $f'(x_0)=df(x_0)/dx \approx (f(x_0+\Delta x)-f(x_0))/\Delta x$ і інтегралу

$F(x_0) = \int_{x_{\min}}^{x_0} f(x)dx \approx \sum_{i=1}^{n} f(x_i)\Delta x$ в заданій точці x_0, де $n = (x_0 - x_{min})/\Delta x$.

21. Створити модуль і пакет з класом з попередньої задачі. Імпортувати модуль і створити об'єкти класу.

22. Створити клас, який описує поняття вектора $\{a_1, a_2, a_3\}$ і містить функції для обчислення його довжини $\sqrt{a_1^2 + a_2^2 + a_3^2}$, додавання двох векторів $c_i = a_i + b_i$ (перевантажити `__add__`), скалярного множення $c = \sum_{i=1}^{3} a_i b_j$ (перевантажити `__mul__`) і векторного множення:

$$c = a \times b = \begin{Bmatrix} a_2 \cdot b_3 - a_3 \cdot b_2 \\ a_3 \cdot b_1 - a_1 \cdot b_3 \\ a_1 \cdot b_2 - a_2 \cdot b_1 \end{Bmatrix}.$$

23. Розв'язати попередню задачу шляхом створення класу-контейнера.

24. Дослідити структуру довільного класу і об'єктів за допомогою функцій `type()`, `dir()`, `vars()`, `getattr()`, методів `__sizeof__()`, `__subclasses__()`, атрибутів `__dict__`, `__doc__`, `__bases__` та модуля `inspect`.

25. Створити і використати декоратор, що повертає похідну функції *f(x)* заданого порядку. Підказка: для першого порядку декоратор повинен повертати нову функцію `f_=lambda x: (f(x+0.001)-f(x))/0.001`.

26. Обчислити декартів добуток множин {1,2,3}, {4,5,6} з використанням оператора `for`. Перевірити результат функцією `product` модуля `itertools`.

27. За допомогою функцій модуля `itertools` отримати усі можливі комбінації і розміщення елементів множини {1, 2, 3} по 2 елементи.

28. Яка дата получиться, якщо від дати 2019-01-02 02:17:01 відняти 1e6 секунд? Який це день тижня?
29. Дано рядок, який містить підрядки <ключ>=<значення>. Зберегти усі ключі і значення у базу даних. Прочитати з бази даних значення за заданим ключем. Використати модулі `anydbm` або `sqlite3`.
30. Зберегти у текстовий файл список елементів поточного каталогу і вкладених каталогів. Створити zip-архів з цим файлом і tar-архів з поточним каталогом. Визначити розмір цих архівів в байтах. Створити новий каталог і скопіювати в нього ці файли.
31. Розв'язати задачу про пошук мінімуму/максимуму функції шляхом поділу області пошуку на дві частини і розпаралелювання пошуку на два процеси. Використати модуль `subprocess`.
32. Розв'язати попередню задачу шляхом використання модуля `multiprocessing`.
33. Розв'язати попередню задачу шляхом використання `concurrent.futures`.
34. Розв'язати попередню задачу шляхом використання `dask`.
35. Розв'язати попередню задачу шляхом виконання одного процесу на локальній машині, а другого – на віддаленій. Використати модуль `socket` або `SocketServer`.
36. Розв'язати попередню задачу шляхом використання `dask.distributed`.
37. Розробити веб-програму для обчислення значення функції *f(x)*. Використати `CGIHTTPServer.`
38. Розробити веб-програму для обчислення значення функції *f(x)*. Використати стандартну `wsgiref.simple_server` або `bottle.`
39. Створити програму, яка отримує дані від веб-сервера (дивись попередні задачі). Використати модулі `urllib2` та `HTMLParser`.
40. Створити програму, яка записує значення функції *f(x)* у XML-документ в такому вигляді: `<point><x>0.1</x><y>-1.25</y></point>`. Використати `xml.dom.minidom`.
41. Створити програму, яка шукає в XML-документі з попередньої задачі значення *y>0*. Використати `xml.etree.ElementTree` або `lxml`.

42. Розробити програму з графічним інтерфейсом Tkinter, яка виводить на вікно документ XML з підсвічуванням синтаксису. Використати `pygments`.

43. Розробити програму з графічним інтерфейсом Tkinter для виведення значення функції *f(x)*. Передбачити обробку виняткових ситуацій, які виникають під час введення недопустимих значень аргументу.

44. Розробити програму з графічним інтерфейсом Tkinter для виведення списку значень функції *f(x)*. Використати такі класи як `Button`, `Label`, `Entry`, `Checkbutton`, `Radiobutton`, `Listbox`.

45. Розробити програму з графічним інтерфейсом PyQt4 або PySide для попередніх задач.

46. Розробити бібліотеку DLL мовою C з функцією, що повертає список значень функції *f(x)*. Викликати цю функцію з модуля Python.

47. Розробити Python-модуль розширення мовою C++, який повертає список значень функції *f(x)*.

48. Створити масив значень функції *f(x)* за допомогою `numpy`. Побудувати графік функції за допомогою `matplotlib`.

49. Розв'язати задачі про пошук коренів рівняння *f(x)=1*, мінімуму функції *f(x)* і її інтегралу з використанням функцій `numpy`.

50. Розв'язати систему лінійних рівнянь за допомогою `numpy.linalg`:

$$\begin{cases} a+b+c=4, \\ 2a-b+2c=2, \\ a+2b-c=4. \end{cases}$$

51. Згенерувати випадкову вибірку з нормального розподілу (μ=20, σ=2, N=1000). Обчислити емпіричні середнє значення та середньоквадратичне відхилення.

52. Дано поліном $3x^2-5x+2$. За допомогою методів класу `numpy.poly1d` обчислити значення полінома, його похідної і первісної в точці x=5.

53. За допомогою scipy-функцій `diff` та `cumtrapz` побудувати графіки похідної і первісної функції *f(x)*.

54. За допомогою `scipy` обчислити визначений інтеграл $\int_{-1}^{1} f(x)dx$.

55. Розв’язати систему диференціальних рівнянь з початковими умовами $x = 2, x' = 1$:

$$\frac{dx}{dt} = x',$$

$$\frac{dx'}{dt} = -x.$$

56. Дано дискретний набір значень x = [0, 1, 2, 3], y = [0, 1, 4, 9]. Обчислити значення y для x=2,5 шляхом лінійної інтерполяції і інтерполяції квадратичним сплайном.

57. За допомогою `scipy.optimize.fsolve` розв’язати рівняння $x^2 - 2x - 2 = 0$.

58. За допомогою `scipy.optimize.root` розв’язати систему нелінійних рівнянь

$$x^2 + 2y = 0,$$
$$x + y^2 = 1.$$

59. За допомогою `scipy.optimize.curve_fit` знайти регресію виду $f(x) = ax^2 + bx + c$ за даними x, y (табл.). Знайти коефіцієнт детермінації R^2.

Таблиця 2 - Емпіричні дані - залежність y від x

x	0	1	2	3
y	0	1	4	10

60. За допомогою `scipy.optimize.curve_fit` знайти регресію виду $f(x, y) = a + bx^2 + cx + dy^2 + ey + fxy$ за даними x, y, z (табл.) Знайти коефіцієнт детермінації R^2.

Таблиця 3 - Емпіричні дані - залежність z від x і y

y \ x	1	2	3	4
1	0	0	1	4
2	0	1	4	8
3	1	4	8	10
4	4	8	10	20

61. Знайти мінімум функції *f(x)* в межах [-1, 1] за допомогою `scipy.optimize`. Спробуйте різні методи оптимізації. Визначте тривалість обчислень для кожного методу.

62. Дано випадкову величину з нормальним розподілом (μ=20, σ=2). Знайти імовірність попадання значень в інтервал (15, 25). Знайти квантілі з рівнем 0,1 і 0,9.
63. Дано випадкову величину з нормальним розподілом (μ=20, σ=2). Згенерувати з неї вибірку розміром 100 значень. Візуально порівняти гістограму вибірки і функцію густини цього розподілу.
64. За допомогою дискретного перетворення Фур'є розрахувати спектр частот для сигналу

$$2\cos(3\pi t) + \cos(5\pi t) + \cos(9\pi t).$$

65. За результатами попередньої задачі і за допомогою оберненого дискретного перетворення Фур’є отримати відфільтрований сигнал з частотами, які більші 7.
66. Обчислити центроїди двох кластерів за даними:

x	5	5	4	6	4	4	5	6	6	4	0	0	1	0	1	1	0	2	2	1
y	5	4	3	6	5	4	4	4	5	5	1	0	0	2	1	2	1	1	0	1

67. За допомогою `xlwt` створити в Excel наступну таблицю:

A	*B*
1	2
2	
3	-1
4	3
5	6

68. Виконати попередню задачу з використанням `win32com.client` і Excel.
69. Зберегти попередню таблицю як файл CSV. За допомогою `pandas` прочитати цей файл і створити об’єкт `DataFrame`. Відкинути рядки з відсутніми даними та з даними *B*<0. Створити новий стовпчик *C* з сумою *A*+*B*. Конвертувати стовпчики *A* і *B* в `numpy.ndarray`.
70. Дано два класи 0 і 1 з ознаками *x* і *y*:

x	5	5	4	6	4	4	5	6	6	4	0	0	1	0	1	1	0	2	2	1
y	5	4	3	6	5	4	4	4	5	5	1	0	0	2	1	2	1	1	0	1
Клас	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

За допомогою scikit-learn визначити клас точки *x*=5, *y*=6.

71. Дано граф, заданий вершинами 1, 2, 3, 4 і ребрами (1, 2), (2, 3), (2, 4), (3, 4). Знайти найкоротший шлях між вузлами 1, 4.
72. Дано предикат "*причина*" та факти "*A причина B*", "*B причина C*", "*A причина D*". Задано правило логічного виведення "*якщо X причина Y, то Y наслідок X*". Знайти усі наслідки *A*. Використати `pyDatalog` або `kanren` або зв'язок з інтерпретатором Prolog.
73. Дано змінну a з множиною допустимих значень $\{1, 3, 5, 7, 9\}$ та змінну b з множиною допустимих значень $\{0, 2, 4, 6, 8\}$. За допомогою `python-constraint` знайти усі розв'язки задачі $a+b>10$.
74. За допомогою PIL відкрити будь-яке зображення, повернути його на 90 градусів вправо і зберегти в відтінках сірого. За допомогою `image.getpixel((x,y))` визначити колір центрального пікселя.
75. За допомогою `pyOpenGL` або `pyglet` створити тривимірну сцену з трьома кубами, які розташовані рядом і повернуті на різні кути.
76. За допомогою `pythonOCC` або `FreeCAD` створити тривимірне тіло шляхом об'єднання призми і сфери. Обчислити об'єм тіла.
77. Створити макрос для Abaqus/CAE для обчислення напружень в осесиметричній моделі циліндричного тіла.
78. Дано вираз $x^2 + sin^2x + cos^2x - 2$. За допомогою `SymPy` виконати підстановку $x = (x + 1)^2$, спростити вираз, отримати похідну і первісну в символьному виді. Обчислити значення виразу, похідної та первісної для x=2.
79. За допомогою `SymPy` розв'язати в символьному виді рівняння відносно x

$$x^2 + 2x - a = 0.$$

80. За допомогою `SymPy` розв'язати в символьному виді систему рівнянь відносно x,y

$$ax - 3y = 2,$$
$$x + 2y = 0.$$

81. За допомогою `sympy.dsolve` розв'язати систему диференціальних рівнянь (знайти функції *x(t), x'(t)*)

$$\frac{dx}{dt} = x',$$
$$\frac{dx'}{dt} = -x.$$

82. За допомогою `OMPython` та OpenModelica розв’язати попереднє рівняння з початковими умовами $x = 2, x' = 1$.

83. За допомогою `win32com.client` змінити розміри довільної параметричної моделі SOLIDWORKS і перебудови її.

84. За допомогою `pyserial` та `pickle` записати і прочитати довільний Python-об’єкт через віртуальні послідовні порти.

85. За допомогою PyParsing розробити синтаксичний аналізатор G-коду для верстату з числовим програмним керуванням. Обмежитись командами `G00` і `G01`. Наприклад:

```
G00 X0.0 Y0.0 Z0.0
G01 X0.0 Y1.0 Z0.0
```

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Allen B. Downey. Think Complexity. - Needham, Massachusetts: Green Tea Press, 2011. - 146p.
2 Allen B. Downey. Think Stats, Second Edition. - O’Reilly, 2015. - 255p.
3 Artificial Intelligence with Python / Prateek Joshi. - Birmingham: Packt, 2017. - 437p.
4 Ashish Kumar. Learning Predictive Analytics with Python - Packt Publishing, 2016. - 493p.
5 Doug Hellmann. Python Module of the Week [Electronic resource]. – Mode of access: https://pymotw.com/2/
6 H. Berendsen. A Student’s Guide to Data and Error Analysis. - Cambridge University Press, 2011. - 239p.

7 H. J. C. Berendsen. Simulating the physical world. Hierarchical Modeling from Quantum Mechanics to Fluid Dynamics. - Cambridge University Press, 2007. - 626p.
8 Hans Petter Langtangen. A Primer on Scientific Programming with Python. 5th Edition. - Springer, 2016. - 942p.
9 Jaan Kiusalaas. Numerical Methods in Engineering with Python. - 2ed. - Cambridge University Press, 2010. - 434p.
10 José Unpingco. Python for Probability, Statistics, and Machine Learning. - Springer, 2016. - 288p.
11 Landau R. H. Computational Physics. Problem Solving with Python, 3rd completely revised edition / Rubin H. Landau, Manuel J. Páez, Cristian C. Bordeianu. - Weinheim: Wiley-VCH, 2015. - 647p.
12 Luca Massaron, Alberto Boschetti. Regression Analysis with Python. - Packt Publishing, 2016. - 416p.
13 Pratik Desai. Python Programming for Arduino. - Birmingham: Packt Publishing, 2015. - 576 p.
14 Scipy Lecture Notes [Electronic resource]. – Mode of access: http://www.scipy-lectures.org/
15 Shane Torbert. Applied Computer Science. Second Edition. - Fairfax: Springer, 2016. - 291p.
16 Swaroop C. H. A Byte of Python [Электронный ресурс]. Пер. с англ. Режим доступа: http://wombat.org.ua/AByteOfPython
17 Thomas Haslwanter. An Introduction to Statistics with Python: With Applications in the Life Sciences. - Springer, 2016. -285p.
18 Андерс М. Написание скриптов для Blender 2.49. Пер. с англ. - PACK, 2010. - 348c.
19 Бизли Д. Python. Подробный справочник. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 864 с., ил.
20 Бринк Хенрик. Машинное обучение / Бринк Хенрик, Ричардс Джозеф, Феверолф Марк. -СПб.: Питер, 2017. -336 с.
21 Буйначев, С. К. Основы программирования на языке Python: учебное пособие / С. К. Буйначев, Н. Ю. Боклаг. – Екатеринбург: Изд-во Урал. ун-та, 2014. – 91 с.
22 Буйначев, С. К. Применение численных методов в математическом моделировании : учебное пособие / С. К.

Буйначев. – Екатеринбург: Издательство Уральского университета, 2014. – 70, [2] с.
23 Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - СПб.: Питер, 2017. - 288 с.: ил.
24 Бэрри, Пол. Изучаем программирование на Python / Пол Бэрри; пер. с англ. — Москва : Издательство «Э», 2017. — 624 с. : ил.
25 Вабищевич П. Н. Численные методы: Вычислительный практикум / Петр Николаевич Вабищевич. — М.: Книжный дом «ЛИБРОКОМ», 2010. — 320 с.
26 Васильев А. Н. Python на примерах. Практический курс · по программированию. - СПб.: Наука и Техника, 2016. - 432 с.: ил.
27 Гифт Н., Джонс Д. Python в системном администрировании UNIX и Linux - Пер. с англ. - СПб.: Символ-Плюс, 2009. - 512 с, ил.
28 Гойвертс Я., Левитан С., Регулярные выражения. Сборник рецептов, 2-е изд. - СПб.: Символ-Плюс, 2015. - 704с.
29 Грас Дж. Data Science. Наука о данных с нуля: Пер. с англ. - СПб.: БХВ-Петербург, 2017. - 336 с.: ил.
30 Гринберг М. Разработка веб-приложений с использованием Flask на языке Python. - М.: ДМК Пресс, 2014. - 272 с.
31 Доля П. Г. Введение в научный Python. Харьков: ХНУ, 2016 - 265с.
32 Доусон М. Программируем на Python. - СПб.:Питер,2014. - 416 с.: ил.
33 Карау Х., Конвински Э., Венделл П., Захария М. Изучаем Spark: молниеносный анализ данных. - М.: ДМК Пресс, 2015. - 304 с.: ил.
34 Копей, В.Б. Ядро геометричного моделювання Open CASCADE Technology для Python-програмістів: Методичні вказівки для самостійної роботи / В.Б. Копей. - Івано-Франківськ : ІФНТУНГ, 2017. - 47с.
35 Луис Педро Коэльо, Вилли Ричарт. Построение систем машинного обучения на языке Python 2-е издание / Пер. с англ. Слинкин А. А. - М.: ДМК Пресс, 2016. - 302 с.: ил.

36 Лутц М. Python. Карманный справочник, 5-е изд.: Пер. с англ. - М.: ООО "И.Д.Вильямс", 2015. - 320 с.: ил.
37 Лутц М. Изучаем Python, 4-е издание. - Пер. с англ. - СПб.: Символ-Плюс, 2011. - 1280 с.: ил.
38 Лучано Ромальо. Python. К вершинам мастерства - М.: ДМК Пресс, 2016. - 768 с.
39 Любанович Билл Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. —480 с.: ил.
40 Марк Саммерфилд. Python на практике. / Пер. с англ. Слинкин А.А. - М.: ДМК Пресс, 2014. - 338 с.: ил.
41 Марманис Х., Бабенко Д. Алгоритмы интеллектуального Интернета. Передовые методики сбора, анализаи обработки данных. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 480 с., ил.
42 Мэтиз Эрик Изучаем Python. Программирование игр, визуализация данных, вебприложения. — СПб.: Питер, 2017. — 496 с.: ил.
43 Мюллер А. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными / Андреас Мюллер, Сара Гвидо. - Вильямс, 2017. - 480с.
44 Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. — СПб.: Питер, 2018. — 576 с.: ил
45 Програмування числових методів мовою Python : навч. посіб. / А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий, Є. В. Глушко; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2013. – 463 с.
46 Рашка С. Python и машинное обучение / пер. с англ. А . В . Логунова. - М.: ДМКПресс, 2017. - 418 с.: ил.
47 Рейтц К., Шлюссер Т. Автостопом по Python. — СПб.: Питер, 2017. — 336 с.: ил.
48 Ричардсон, Крэйг. Программируем с Minecraft. Создай свой мир с помощью Python / Крэйг Ричардсон; пер. с англ. —М.: Манн, Иванов и Фербер, 2017. — 368 с.: ил.
49 Свейгарт, Эл. Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих. - М.: Вильямс, 2017. - 592с.

50 Сегаран. Т. Программируем коллективный разум. – Пер. с англ. – СПб: Символ-Плюс, 2008. – 368 с., ил.
51 Сейтц Джастин. Программирование на Python для хакеров, Пер. с англ. 2012. - 208с.
52 Силен Д. Основы Data Science и Big Data. Python и наука о данных / Силен Дэви, Мейсман Арно, Али Мохамед - СПб.: Питер, 2017. - 336с.
53 Соловьёв И. А., Червяков А. В., Репин А. Ю. Вычислительная математика на смартфонах, коммуникаторах и ноутбуках с использованием программных сред Python: Учебное пособие. — СПб.: Издательство «Лань», 2011. — 272 с: ил.
54 У. Сэнд, К. Сэнд. Hello World! Занимательное программирование. — СПб.: Питер, 2016. — 400 с.: ил.
55 Уэс Маккинли. Python и анализ данных/ Пер. с англ. Слинкин А. А. - М.: ДМК Пресс, 2015. - 482 с.: ил.
56 Федоров Д. Ю. Основы программирования на примере языка Python : учеб.пособие / Д. Ю. Федоров. – СПб., 2016. – 176 с.
57 Шоу, Зед. Легкий способ выучить Python / Зед Шоу; пер. с англ. — Москва : Издательство «Э», 2017. — 352 с.
58 Язык программирования Python. / Г.Россум, Ф.Л.Дж.Дрейк, Д.С.Откидач, М.Задка, М.Левис, С.Монтаро, Э.С.Реймонд, А.М.Кучлинг, М.-А.Лембург, К.-П.Йи, Д.Ксиллаг, Х.Г.Петрилли, Б.А.Варсав, Дж.К.Ахлстром, Дж.Роскинд, Н.Шеменор, С.Мулендер. – 2001. - 454с.